UNIVERZA V LJUBLJANI
FAKULTETA ZA RAČUNALNIŠTVO IN INFORMATIKO

Aleš Kern

Prilagoditev obstoječih poročil novi verziji sistema Microsoft Dynamics NAV

DIPLOMSKO DELO
NA UNIVERZITETNEM ŠTUDIJU

Mentor: prof. dr. Viljan Mahnič

Ljubljana, 2016

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Naslov naloge:

Prilagoditev obstoječih poročil novi verziji sistema Microsoft Dynamics NAV

Tematika naloge:

Z uvedbo tronivojske arhitekture, na kateri temelji verzija Microsoft Dynamics NAV 2013, se je bistveno spremenil način kreiranja poročil. Organizacije, ki so uporabljale starejše verzije omenjenega sistema, bi morale praktično vsa poročila kreirati na novo. Temu se lahko izognemo z orodjem, ki omogoča (skoraj) avtomatsko pretvorbo poročil iz starega klasičnega formata v novo obliko. Orodja, ki jih v ta namen ponuja Microsoft, pa so pomanjkljiva in zahtevajo veliko ročnega dela.

V diplomski nalogi realizirajte orodje za pretvorbo poročil, ki so bila kreirana v starem formatu, primernem za dvonivojsko arhitekturo, v nov RDLC format, ki ga zahtevajo Microsoft Dynamics NAV 2013 in novejše verzije. V ta namen proučite in podrobno opišite postopek kreiranja poročil v obeh formatih, izdelajte načrt programske rešitve in jo realizirajte. Kakovost rešitve primerjajte z orodji, ki jih v ta namen ponuja Microsoft.

IZJAVA O AVTORSTVU

diplomskega dela

Spodaj podpisani/-a Aleš Kern,

z vpisno številko 63060109,

sem avtor/-ica diplomskega dela z naslovom:

Prilagoditev obstoječih poročil novi verziji sistema Microsoft Dynamics NAV

S svojim podpisom zagotavljam, da:

- sem diplomsko delo izdelal/-a samostojno pod mentorstvom
  prof. dr. Viljana Mahniča
- so elektronska oblika diplomskega dela, naslov (slov., angl.), povzetek (slov., angl.) ter ključne besede (slov., angl.) identični s tiskano obliko diplomskega dela
- soglašam z javno objavo elektronske oblike diplomskega dela v zbirki "Dela FRI".

V Ljubljani, dne 7. julija 2016 Podpis avtorja/-ice:

Zahvala

Zahvalil bi se svoji družini, ki mi je v času pisanja diplomske naloge stala ob strani, pa tudi prijateljem, sošolcem in sodelavcem, ki so me pri tem spodbujali. Za pridobljeno znanje bi se zahvalil vsem profesorjem na FRI-ju, posebej pa še mentorju prof. dr. Viljanu Mahniču, za vse nasvete in pomoč pri pisanju diplomske naloge.

Kazalo

Seznam uporabljenih kratic in simbolov

RDLC (ang. Report Definition Language Client-side) – Microsoft standard za prikaz poročil. Dejansko procesiranje in vizualizacija se izvaja na strani odjemalca.

ERP (ang. Enterprise Resource Planning) – Integrirani poslovni informacijski sistem.

SUPB (ang. DMBS – Database Management System) – Sistem za upravljanje s podatkovno bazo.

RTC (ang. Role Tailored Client) – Uporabniški vmesnik, preko katerega deluje povezovanje na Microsoft Dynamics NAV v tronivojski arhitekturi.

WS (ang. Web Service) – Je storitev, s katero dve elektronski napravi medsebojno komunicirata z uporabo spleta.

XML (ang. Extensible Markup Language) – Označevalni jezik, ki nam omogoča strukturiran opis podatkov.

FOB (ang. Financials Object file) – Binarna datoteka, ki vsebuje enega ali več Dynamics NAV objektov.

SQL (ang. Structured Query Language) – Programski jezik za delo s podatki v relacijskih podatkovnih bazah.

C/AL (ang. Client Aplication Language) – Programski jezik za pisanje kode v Microsoft Dynamics NAV.

Povzetek

V sklopu tega diplomskega dela smo razvili orodje, ki omogoča lažjo pretvorbo poročil pri procesu nadgradnje informacijskega sistema Microsoft Dynamics NAV. Nadgradnja poročil iz klasičnega v RDLC način poročanja je kompleksen in časovno potraten postopek. Predstavljen je informacijski sistem Microsoft Dynamics NAV, osnove razvoja ter klasični in RLDC način poročanja v njem. Pri tem so podrobno popisane razlike v kreaciji in vizualizaciji poročil pri obeh sistemih poročanja. Opisan je postopek nadgradnje poročil z obstoječimi orodji za nadgradnjo. Opravljena je tudi analiza prostorske kompleksnosti ploskega nabora podatkov. Pri razvoju programske rešitve je bil fokus predvsem v pretvorbi vizualizacije. Z razvojem orodja smo želeli omogočiti lažjo pretvorbo poročil iz klasičnega v RLDC sistem poročanja in s tem posledično lažjo in hitrejšo nadgradnjo informacijskega sistema Microsoft Dynamics NAV. Naredili smo tudi primerjavo rezultatov programske rešitve, razvite v sklopu diplomskega dela, z rezultati že obstoječih rešitev.

Ključne besede:

poročanje, nadgradnja, Dynamics NAV, pretvorba, ploski nabor podatkov, RDLC, vizualizacija

Abstract

A tool was developed, which can be used for partial automatic upgrade of reports in the upgrade process of Microsoft Dynamics NAV. The upgrading of reports from classic to RDLC system of reporting is a complex and time consuming process. The basics of development in Microsoft Dynamics NAV are described as is the classic reporting and RDLC reporting, that was introduced with the adoption of three-tiered architecture. The changes in report creation and visualization of reports in both reporting systems are described. The spacial complexity of flat dataset is also analysed. Described is also the report upgrading process with existing tools. Our focus is primarily on how visualization must be upgraded. The tool that we have developed seeks to eliminate the most time consuming aspects of report upgrading and offers a faster and easier way of upgrade for Microsoft Dynamics NAV. The results of solution, developed in the content of this work, are compared with results from existing solutions.

Key words:

reporting, upgrade, Dynamics NAV, optimization, flat dataset, RDLC, visualization

Poglavje 1

Uvod

V današnji informacijski dobi so informacijski sistemi postali nepogrešljivi del praktično vsakega podjetja. Informacijski sistemi pa se skozi čas razvijajo, spreminjajo in dopolnjujejo. Obstoječe verzije je torej treba periodično nadgrajevati, kar pa je lahko tudi težavno. Zaradi sprememb je namreč treba vložiti čas in trud v prenovo obstoječih rešitev, pri čemer pa se končni izdelek po izgledu, funkcionalnosti in namenu včasih bistveno ne razlikuje od začetnega.

V diplomski nalogi obravnavamo primer takšne nadgradnje, in sicer nadgradnjo poročil v informacijskem sistemu Microsoft Dynamics NAV s klasičnim poročanjem na poročila z RDLC (ang. Report Definition Language Client-side) poročanjem. V novejših verzijah Microsoft Dynamics NAV je klasično poročanje nadomeščeno z RDLC poročanjem. Veliko podjetij uporablja starejše verzije Microsoft Dynamics NAV s klasičnim poročanjem in so pri nadgradnji infomacijskega sistema na novo verzijo prisiljena v uvedbo RDLC poročanja. Vsa poročila, razvita za klasično poročanje, je treba pretvoriti ali ponovno razviti, pri čemer so obstoječe rešitve za pretvorbo bodisi plačljive bodisi komplekse in časovno zamudne. Na voljo torej nimamo dobrega orodja, ki bi nam olajšalo nadgradnjo poročil. Ker je razvoj poročil kompleksen in lahko za razvoj enega poročila porabimo tudi več dni, podjetja pa imajo tipično razvite po desetine poročil, predstavlja to kar velik problem pri nadgradnji.

V sklopu te diplomske naloge smo razvili in predstavili programsko rešitev, ki nam z delno avtomatizacijo olajša nadgradnjo poročil med obema sistemoma poročanja. Programska rešitev nam vhodno tekstovno datoteko, izvoženo iz Microsoft Dynamics NAV s klasičnim poročanjem, pretvori v obliko, primerno za uvoz v verzije Microsoft Dynamics NAV z RDLC sistemom poročanja.

V nadaljevanju diplomske naloge bomo v poglavju 2 predstavili razvojno

okolje Microsoft Dynamics NAV, oba sistema poročanja pa tudi ozadje in razloge, ki so pripeljali do spremembe sistema poročanja. Pregledali bomo tudi časovno zahtevnost ploskega nabora podatkov ter našteli možnosti za njegovo optimizacijo. V poglavju 3 bomo opisali našo programsko rešitev z arhitekturnega vidika in popisali potrebne transformacije, ki jih vršimo pri pretvorbi. V poglavju 4 bomo primerjali rezultate naše programske rešitve s poročili, pretvorjenimi s pomočjo drugih orodij, ter z ročno pretvorjenimi poročili. Pregledali bomo tudi, kako bi se lahko programsko rešitev še dodatno izboljšalo. V poglavju 5 pa bomo povzeli zaključke te diplomske naloge.

Poglavje 2

Sistem poročanja v Microsoft Dynamics NAV

2.1 Predstavitev informacijskega sistema Microsoft Dynamics NAV

Microsoft Dynamics NAV je bil prvotno proizvod danskega podjetja Navision A/S, ki ga je Microsoft kupil leta 2002 [6]. Skupaj z ostalimi ERP (ang. Enterprise Resource Planning) produkti je del Microsoft Dynamics aplikacij. V času svojega obstoja se mu je naziv večkrat spremenil. V Sloveniji je najbolj poznan pod imenom Navision. V sklopu te diplomske naloge bomo zanj uporabljali ime Microsoft Dynamics NAV ali samo Dynamics NAV.
Microsoft Dynamics NAV je informacijski sistem, namenjen predvsem za manjša in srednje velika podjetja. Podpira računovodske in finančne potrebe podjetij ter tudi ostala področja, npr. dobavo, proizvodnjo in storitve. Zaradi svoje modularne narave je izjemno razširljiv in sposoben prilagoditi se specifikam posamezne stranke [8].
V letu 2015 ga je uporabljalo kar 110.000 podjetij po vsem svetu, kar predstavlja porast za 8.000 glede na leto 2014 [5].
Microsoft Dynamics NAV je sprva omogočal delo z dvema vrstama sistemov za upravljanje s podatkovno bazo (SUBP):

- domorodni podatkovni strežnik (ang. Native database server)
- Microsoft SQL podatkovni strežnik

Pri domorodnih podatkovnih strežnikih je šlo za povezavo na strežniško FBD (ang. Financials DataBase) datoteko [9]. Od verzije Dynamics NAV 2013 ta

vrsta povezave ni več mogoča [11].
Verziji Dynamics NAV 2009 in Dynamics NAV 2013 sta bili z vidika nadgradnje resnično prelomni. Z verzijo Dynamics NAV 2009 je bila namreč poleg klasične dvonivojske arhitekture uvedena tudi uporabniško prilagojena arhitektura (ang. RoleTailored Architecture), ki je tronivojska. Uveden je bil tudi nov uporabniški vmesnik – uporabniško prilagojen odjemalec (ang. RoleTailored Client - RTC). Od verzije Dynamics NAV 2013 naprej pa povezovanje preko dvonivojske arhitekture ni več možno [2].

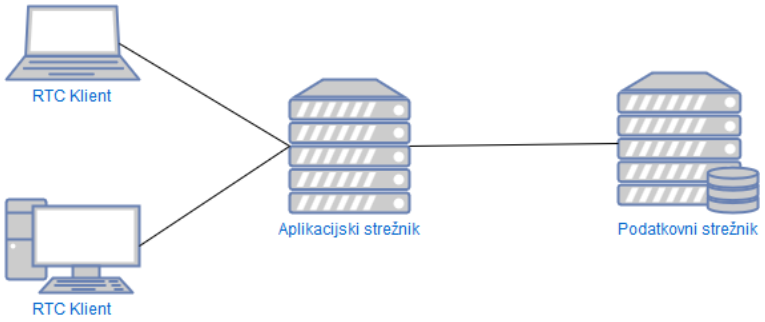
Dvonivojska arhitektura je sestavljena iz dveh delov:

- podatkovni strežnik (domorodni ali Microsoft SQL)
- odjemalec

Tronivojska arhitektura pa je sestavljena iz treh delov [10]:

- podatkovni strežnik (Microsoft SQL podatkovni strežnik)
- aplikacijski strežnik (ang. Microsoft Dynamics NAV server)
- odjemalec (RTC, WS (ang. Web Service) odjemalec ...)

Bistvena razlika med dvonivojsko in tronivojsko arhitekturo je, da je pri tronivojski arhitekturi prisoten tudi aplikacijski strežnik, na katerem se izvaja poslovna logika. Odjemalec tako služi le prikazovanju rezultatov. Pri dvonivojski arhitekturi pa se je poslovna logika izvajala na odjemalčevi strani.



Slika 2.1: Primer tronivojske arhitekture

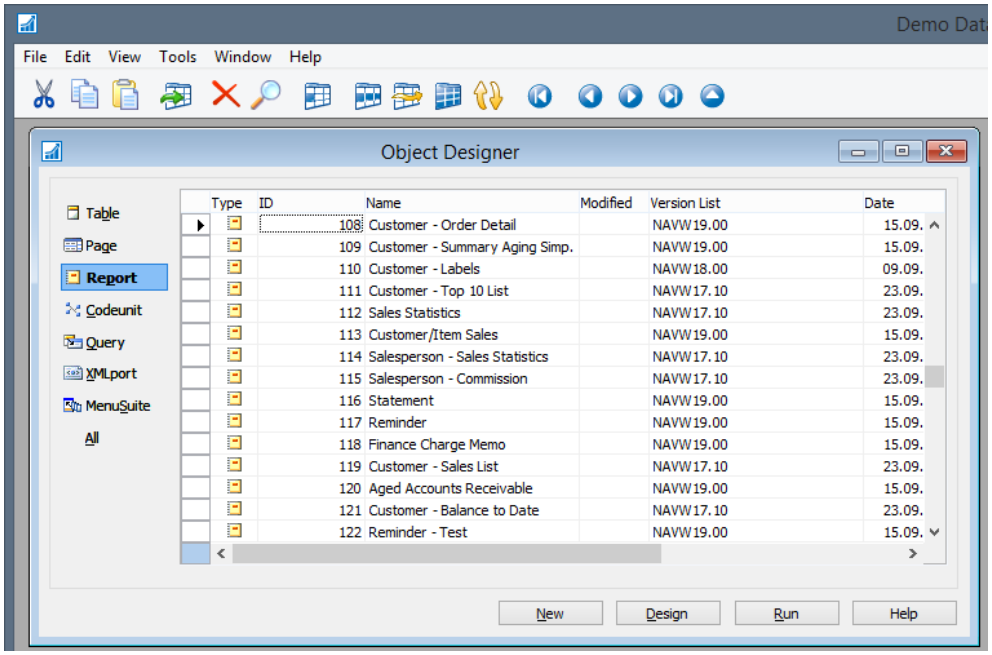
Z uvedbo tronivojske arhitekture se je spremenil tudi sistem poročanja. Pri dvonivojski arhitekturi smo imeli podprt samo klasičen sistem poročanja, pri katerem se je poslovna logika delno lahko izvajala tudi na strani klienta,

ob samem generiranju izpisa. To v tronivojski arhitekturi ni več možno. S tronivojsko arhitekturo je uveden RDLC način poročanja, ki je v njej tudi edini podprt. Od verzije Microsoft Dynamics NAV 2013 ostaja podprt edino RDLC način poročanja.

2.2 Programiranje v Microsoft Dynamics NAV

2.2.1 Razvojno okolje

Programiranje v Microsoft Dynamics NAV poteka v C/SIDE (ang. Client Server Integrated Development Enviroment) okolju. Okolje C/SIDE ni objektno usmerjeno, je pa objektno zasnovano (ang. Object based). Programska koda je napisana v jeziku C/AL (ang. Client Aplication Language), ki temelji na jeziku Pascal.
Dynamics NAV ima vnaprej določene vrste objektov, preko katerih se lahko implementira posamezne elemente uporabniškega vmesnika in poslovne logike. Razvoj poteka preko načrtovalca objektov (ang. Object Designer), glede na tip objekta se nato pri razvoju uporablja enega izmed načrtovalcev (načrtovalec tabel, poročil ...).



Slika 2.2: Prikaz razvojnega okolja in načrtovalca objektov

2.2.2 Vrste objektov

Vrste objektov v Microsoft Dynamics NAV so naslednje [7]:

- Tabela (ang. Table) – je osnovni objekt za delo s podatki. Definira, kako so podatki shranjeni v bazi. Poleg tabel, ki so uporabniško definirane in definirajo podatke v podatkovni bazi, poznamo tudi virtualne tabele, v katerih so shranjeni sistemski podatki in iz katerih se lahko samo bere. Primer virtualne tabele sta npr. tabela. *Integer*, v kateri so shranjena cela števila, in tabela *Date*, v kateri so shranjeni datumi.

- Obrazec (ang. Form) – služi za prikazovanje podatkov uporabniku v dvonivojski arhitekturi. Preko obrazca je možno podatke tudi spreminjati, jih brisati ali dodajati nove. Obrazec se je uporabljal samo do verzije Dynamics NAV 2013, z uvedbo tronivojske arhitekture pa je nadomeščen s stranjo.

- Stran (ang. Page) – služi za prikazovanje podatkov uporabniku v trinivojski arhitekturi. Preko strani je možno podatke tudi spreminjati, jih brisati ali dodajati nove. Prvič se je pojavila v verziji Dynamics NAV 2009. Od verzije Dynamics NAV 2013 nadomešča obrazce.

- Poročilo (ang. Report) – služi za obdelavo večjih količin podatkov ter predstavitvi in tiskanju podatkov (npr. računov, bilanc). Z uvedbo tronivojske arhitekture je prišlo do sprememb v poročilih, saj je bilo klasično poročanje z dvonivojske arhitekture nadomeščeno z RLDC poročanjem.

- Dataport – služi uvozu ter izvozu podatkov v tekstovni obliki. Dataport se je uporabljal samo do verzije Dynamics NAV 2013, od takrat dalje je nadomeščen z XMLportom.

- XMLport – služi uvozu ter izvozu podatkov v XML (ang. Extensible Markup Language) obliki. Od verzije Dynamics NAV 2013 se lahko preko njega uvaža tudi podatke v tekstovni obliki.

- Funkcijska koda (ang. Codeunit)– služi implementaciji poslovne logike nad podatki ob določenih dogodkih (npr. knjiženje računov).

- Meni (ang. Menusuite) – omogoča dodajanje in združevanje bližnjic do ostalih objektov v uporabniško dostopen meni.

- Poizvedba (ang. Query) – omogoča neposredne poizvedbe nad bazo z Microsoft SQL podatkovnim strežnikom. Uvedena je bila z verzijo Dynamics NAV 2013.

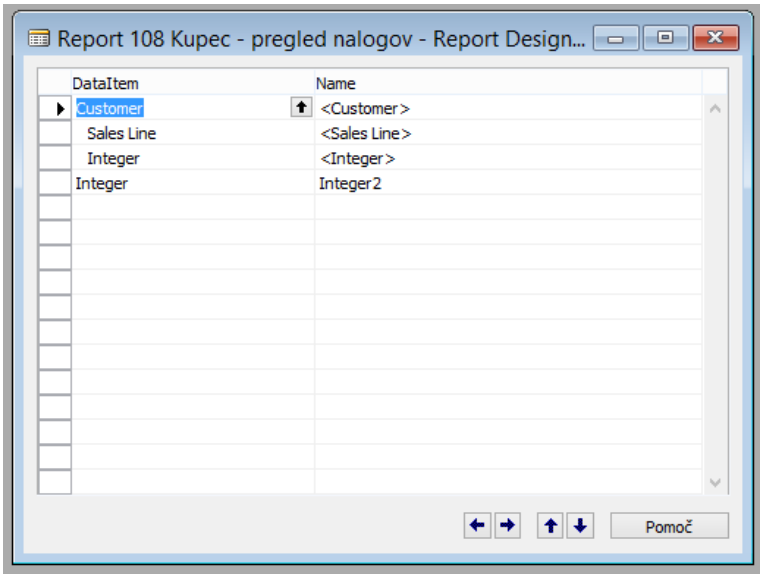
2.2.3 Izvozi ter uvozi objektov

Microsoft Dynamics NAV omogoča izvoze in uvoze objektov, ki so možni za en objekt ali za več objektov skupaj. Kot format izvoza se lahko izbere tekstovna datoteka, XML datoteka (samo poročila, obrazci in strani v Dynamics NAV 2009) ali FOB (ang. Financials Object file) datoteka. V tekstovni datoteki je izvorna C/AL programska koda, medtem ko gre pri izvozu FOB datotek za že prevedeno kodo. XML datoteke se uporabljajo samo pri pretvorbi objektov s pomočjo Microsoftovih orodij za pretvorbo. Več o tem v poglavju 2.4.1.

2.3 Predstavitev sistema poročanja

2.3.1 Klasični način poročanja (poročanje v dvonivojski arhitekturi)

Za poročanje se uporablja objekte tipa poročilo. Implementacija poročil poteka skozi več načrtovalcev. Prvi med njimi je načrtovalec poročil (ang. Report Designer).



Slika 2.3: Primer načrtovalca poročil

Skozi načrtovalca poročil določimo podatkovne elemente (ang. DataItem). Z njimi dostopamo do zapisov tabel, katerih podatki nas zanimajo ter bi jih radi na poročilu prikazali. Vsak podatkovni element ima tudi svoje lastnosti, s

katerimi se lahko določi npr. morebitne filtre nad tabelo, povezavo z ostalimi podatkovnimi elementi, največje število ponovitev podatkov itd.

Podatkovni elementi so urejeni hierarhično. Pri izvajanju poročila se najprej začne izvajati podatkovni element na prvem nivoju. Za vsak najden zapis v njem se nato izvede njemu podrejene podatkovne elemente. V primeru, da za ta podatkovni element ni več najdenih zapisov, se začne izvajati naslednji podatkovni element na istem nivoju (z istim nadrejenim podatkovnim elementom).
Primer izvajanja na poročilu v sliki 2.3 bi bil takšen:

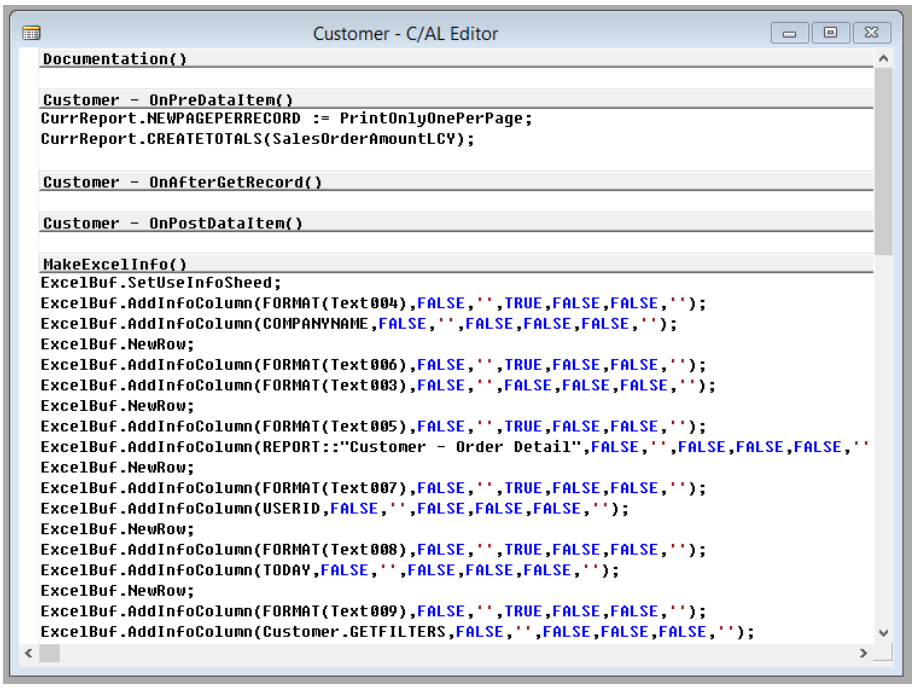
V tabeli kupcev imamo dva kupca: K1 in K2. Vsak kupec ima svoje prodajne vrstice. Podatkovna elementa *Integer* in *Integer2* bereta podatke iz virtualne tabele *Integer*, v kateri se nahaja seznam celih števil. Na poročilu sta s filtri omejena, tako da se podatkovni element *Integer* izvede za število različnih valut v vrsticah kupca, podatkovni element *Integer2* pa samo enkrat.
1. Najde se kupec K1.
1.1. Najde se vse prodajne vrstice za kupca K1.
1.2. Za vsako valuto na prodajnih vrsticah kupca K1 se izračuna seštevek.
2. Najde se kupec K2.
2.1. Najde se vse prodajne vrstice za kupca K2.
2.2. Za vsako valuto na prodajnih vrsticah kupca K2 se izračuna seštevek.
3. Izračuna se seštevke vrstic za vse valute za oba kupca.

Programska koda v poročilih se ureja s pomočjo urejevalnika C/AL (ang. C/AL Editor). Do njega lahko dostopamo preko načrtovalca poročil.
Vsak podatkovni element ima prednastavljene sprožilce (ang. triggers):

- *OnPreDataItem*: koda v njem se izvede pred izvajanjem podatkovnega elementa.
- *OnAfterGetRecord*: koda v njem se izvede po vsakem najdenem zapisu v tem podatkovnem elementu.
- *OnPostDataItem*: koda v njem se izvede po izvajanju podatkovnega elementa.

Podobni sprožilci so tudi drugje, npr. na celotnem poročilu, na obrazcu zahtev (ang. Request Form). Poleg kode, ki jo lahko pišemo na prednastavljene spro-

žilce, pa lahko definiramo poljubne funkcije, ki jih nato kličemo iz sprožilcev. Takšen primer predstavlja funkcija *MakeExcelInfo* na sliki 2.4.



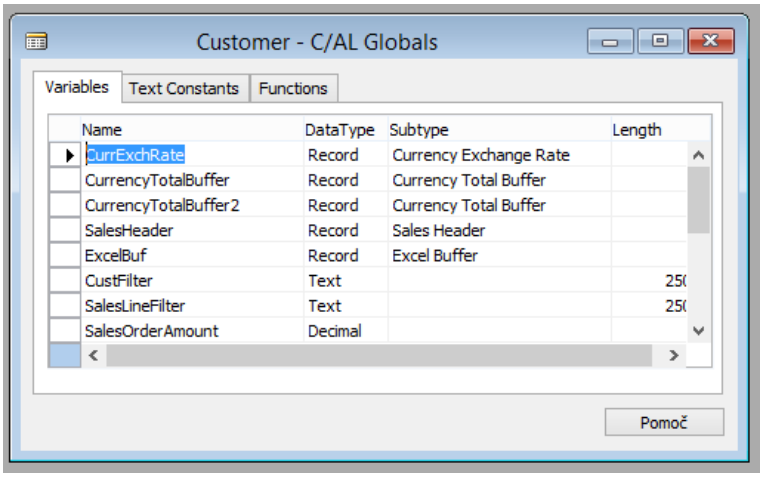
Slika 2.4: Primer urejevalnika C/AL

Za deklaracijo funkcij pa tudi drugih spremenljivk, s katerimi si želimo ob izvajanju poročila pomagati, v poročilu lahko določimo globalne ali lokalne C/AL spremenljivke. Primer je prikazan na sliki 2.5.
Te so lahko tipa:

- Spremenljivke (ang. Variables) – globalne so vidne na ravni celega poročila, lokalne pa na ravni posameznega sprožilca ali funkcije. Spremenljivke so lahko tudi druge tabele, logiko za branje ali pisanje v te tabele je treba razviti.

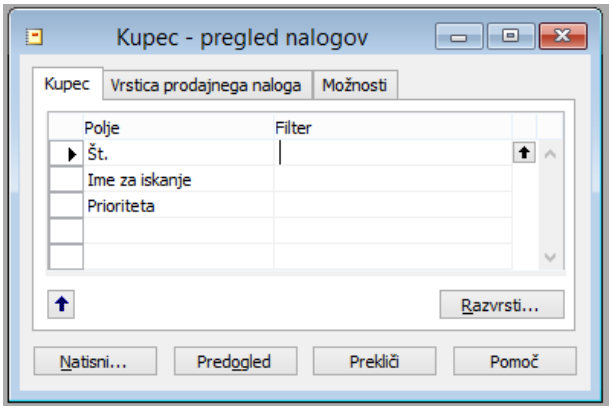
- Tekstovne konstante (ang. Text Constants) – se uporabljajo za prikaz besedila na poročilih. Omogočajo določitev besedila za različne jezike (recimo prvi kupec je tujec in želi besedilo v angleščini, drugi kupec želi slovensko besedilo).

- Funkcije (ang. Functions) – v poročilo lahko dodajamo poljubne funkcije. Na voljo so samo pri globalnih spremenljivkah.



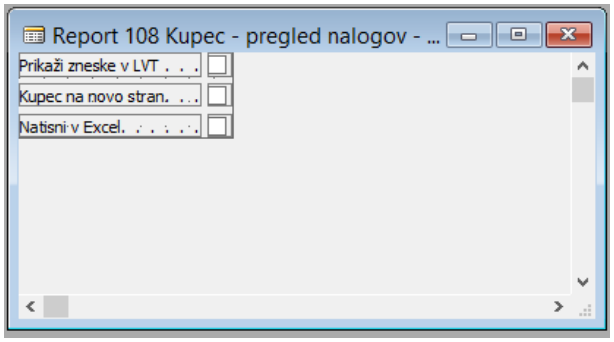
Slika 2.5: Primer globalnih C/AL spremenljivk

Pri zagonu poročila se uporabniku najprej prikaže obrazec zahtev (ang. Request Form). Preko njega ima uporabnik možnost nastaviti filtre na posameznih poljih v tabeli, na katero se povezujemo s podatkovnim elementom. Primer je prikazan na sliki 2.6.



Slika 2.6: Primer obrazca zahtev pri zagonu poročila

Uporabniku pa se lahko ponudijo dodatne možnosti. Te možnosti dodamo na poročilo s pomočjo načrtovalca možnosti obrazca zahtev (ang. Request Options Form Designer), prikazanega na sliki 2.7.



Slika 2.7: Primer načrtovalca možnosti obrazca zahtev

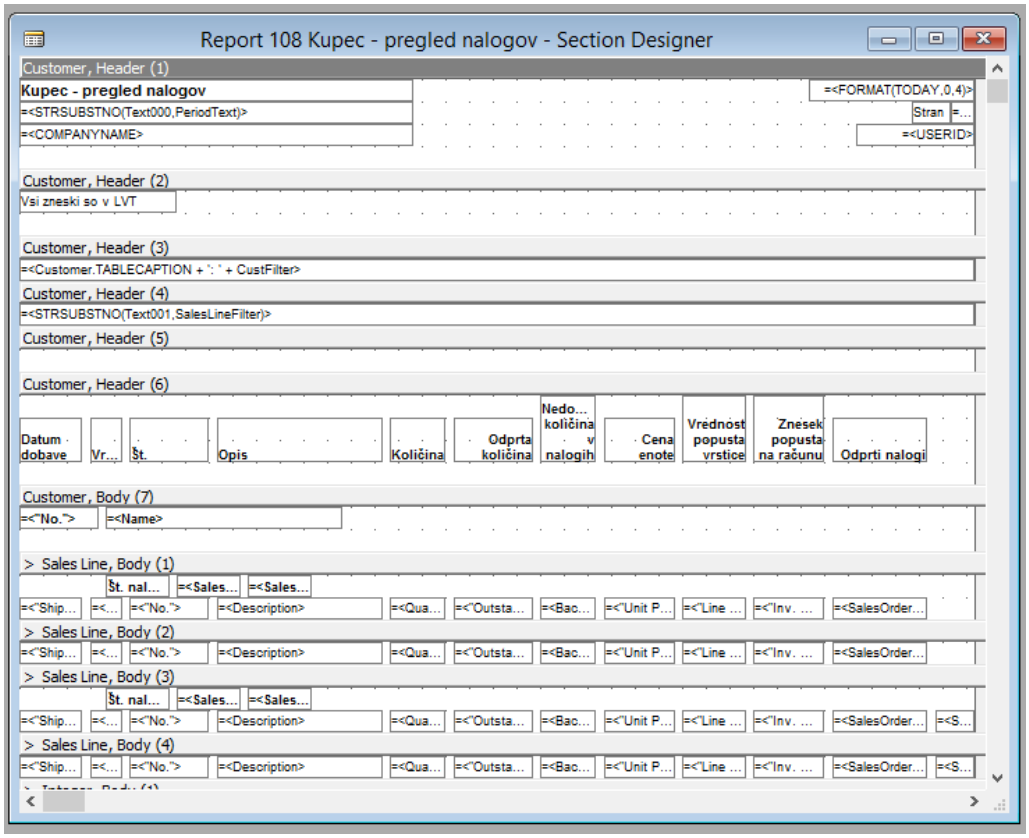
2.3.2 Klasična postavitev poročila

Klasična postavitev poročila (ang. Classic Report Layout) je sestavljena iz sekcij. Te so povezane na posamezen podatkovni element v poročilu. Že ob izvajanju poročila se iz posameznih sekcij generira prikaz poročila.

Izpis poročila lahko natisnemo s poljubnim tiskalnikom, možen pa je tudi predogled poročila. Na predogledu lahko že sredi izvajanja poročila vidimo izpis za že obdelane podatke. Sekcije lahko urejamo z načrtovalcem sekcij (ang. Section Designer). Primer načrtovalca sekcij je viden na sliki 2.8.

Tipi sekcij so vnaprej predpisani. Tip vpliva na obnašanje sekcije pri izpisu poročila. Poznamo naslednje tipe:

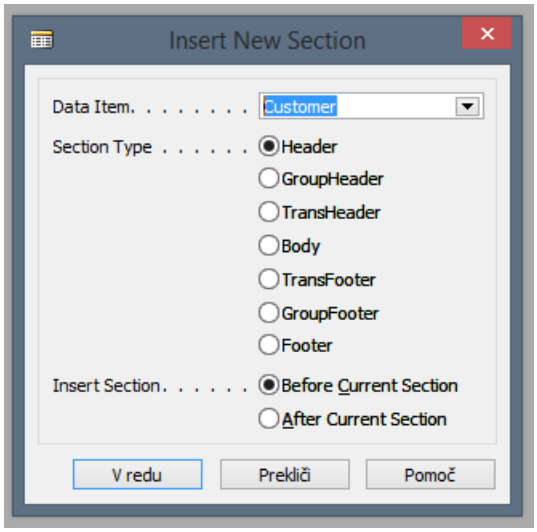
- Glava (ang. Header) – glava podatkovnega elementa se praviloma pojavi enkrat (ko najdemo prvi zapis v tabeli). Na lastnostih sekcije se lahko nastavi, da se sekcija ponavlja na vsaki novi strani.
- Skupinska glava (ang. GroupHeader) – na podatkovnem elementu lahko določimo, da zapise iz tabele grupiramo (npr. grupiramo kupce po državah). Skupinska glava se pojavi na začetku vsake skupine.
- Prehodna glava (ang. TransHeader) – v primeru, da je izpis na več straneh, se prehodna glava pojavlja na novi strani.
- Telo (ang. Body) – sekcija se pojavi za vsak najden zapis v tabeli.
- Prehodna noga (ang. TransFooter) – v primeru, da je izpis na več straneh, se prehodna noga pojavlja na prejšnji strani.
- Skupinska noga (ang. GroupFooter) – na podatkovnem elementu lahko določimo grupiranje zapisov iz tabele (npr. kupce po državah). Skupinska noga se pojavi na koncu vsake skupine.



Slika 2.8: Primer načrtovalca sekcij

- Noga (ang. Footer) – noga podatkovnega elementu se praviloma pojavi enkrat (ko najdemo zadnji zapis v tabeli). Na lastnostih sekcije se lahko nastavi, da se ponavlja na vsaki novi strani.

Sekcije lahko na poročilo poljubno dodajamo, spreminjamo in odstranjujemo. Pri dodajanju izberemo, za kateri podatkovni element želimo dodati sekcijo, katerega tipa naj bo ter, ali naj se sekcija vrine pred ali za trenutno izbrano sekcijo.



Slika 2.9: Primer okna pri dodajanju sekcije

Na sekcije se lahko dodaja kontrole, to so gradniki, s katerimi prikažemo določene podatke na izpisu. Dovoljeni so naslednji tipi kontrol:

- Tekstovno polje (ang. Textbox) – kontrola, s katero prikazujemo alfanumerične znake. Prikazuje vsebino lastnosti Izvor (ang. SourceExpr), ki je lahko polje na tabeli, spremenljivka, tekstovna konstanta, lahko pa Izvor določimo kar na sami kontroli.

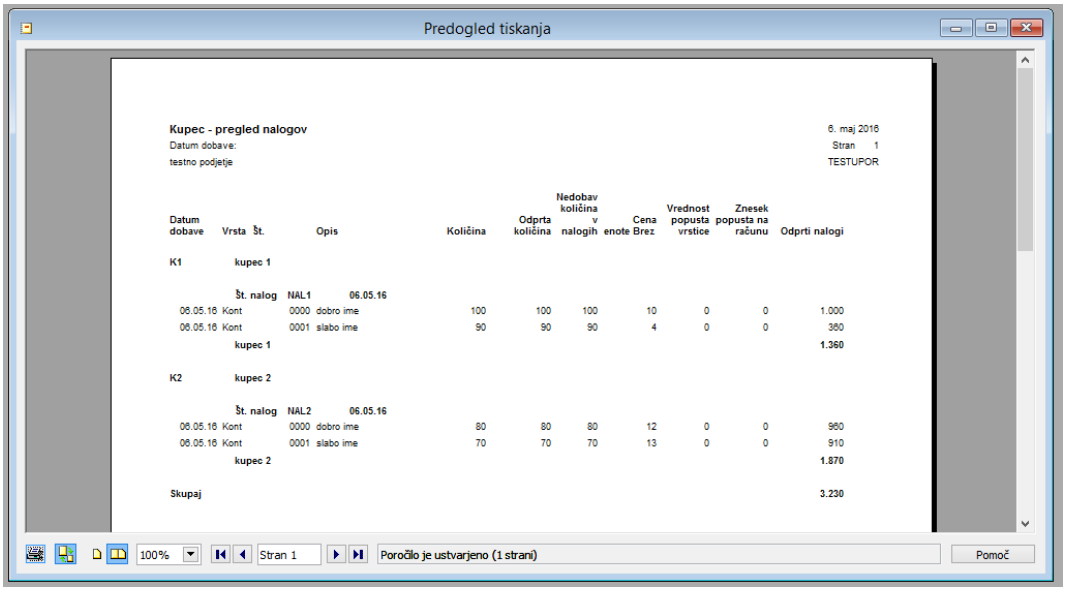
- Oznaka (ang. Label) – prav tako služi za prikaz alfanumeričnih znakov. Prikazuje vsebino lastnosti napis (ang. Caption). Lahko so definirani tudi napisi v različnih jezikih. Oznaka ima lahko definiranega očeta (ang. Parent), od katerega prevzame lastnost napis. Napis pa lahko določimo kar na sami kontroli.

- Slikovno polje (ang. PictureBox) – kontrola za prikaz slik.

- Oblika (ang. Shape) – kontrola, s katero lahko na poročilu prikažemo oblike. Gre za razne pravokotnike in črte.

Pri generiranju izpisa se sekcije obnašajo hierarhično, podobno kot podatkovni elementi. Vsaka sekcija ima tudi svoje sprožilce – *OnPreSection*, *OnPostSection*, na katerih je lahko poljubna koda. Tako se lahko na sprožilcu nahaja tudi klic *CurrReport.SHOWOUTPUT(parameter)*, ki glede na parameter določa, ali se posamezna sekcija na poročilu izpiše ali ne.

Pri izpisu poročila se glede na nastavitve tiskanja avtomatsko razporedi sekcije po straneh. Če je npr. preveč sekcij tipa telo za izpis na eno stran, se izpiše na prvi strani še morebitna sekcija tipa prehodna noga ter toliko sekcij tipa telo, kot jih je možno prikazati na strani. Na novi strani se izpiše sekcija tipa prehodna glava ter preostale sekcije tipa telo.



Slika 2.10: Primer predogleda poročila

2.3.3 Poročanje RDLC (poročanje v tronivojski arhitekturi)

RDLC poročila implementiramo s pomočjo načrtovalca nabora podatkov poročil (ang. Report Dataset Designer), ki je prikazan na sliki 2.11. Ta se razlikuje od načrtovalca poročil v klasičnem poročanju v tem, da ne določa samo podatkovnih elementov, ampak tudi vse podatke, ki jih želimo prikazati na poročilu oziroma so potrebni za izgradnjo postavitve poročila. Zaradi uvedbe tronivojske arhitekture se je namreč spremenil način izpisa poročil. Ali gre za podatkovni element ali podatek, je podano z lastnostjo podatkovni tip (ang. Data Type). Ta je v primeru podatkov stolpec (ang. Column).

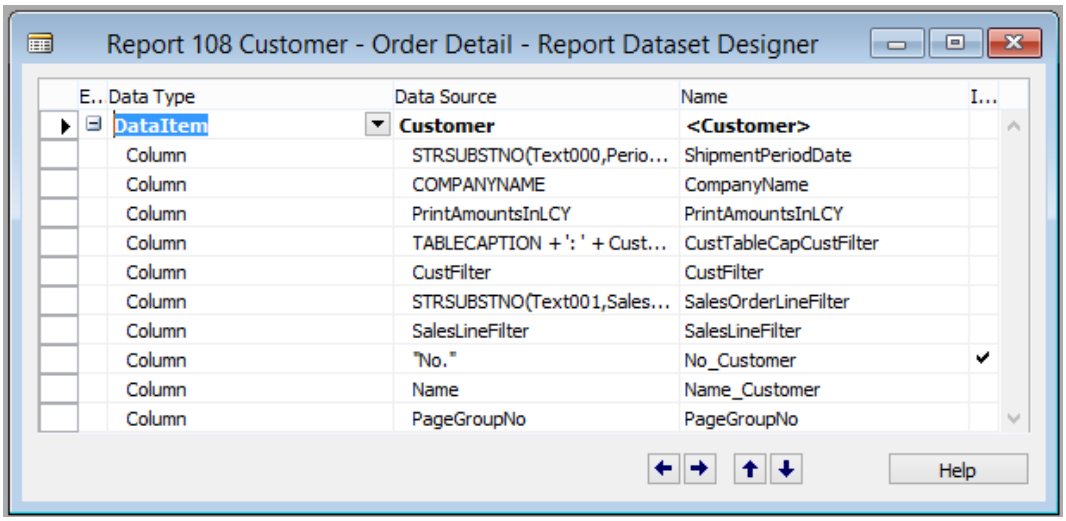
Sam izpis poročil se izvaja na odjemalcu, medtem ko se programska koda izvaja na strani aplikacijskega strežnika. Naloga aplikacijskega strežnika je torej, da izvede programsko kodo, na podlagi nje pripravi podatke ter jih po-

sreduje odjemalcu. Naloga odjemalca pa je, da posredovane podatke pravilno vizualizira.

Največja razlika v primerjavi s klasičnim poročanjem je vloga, ki jo opravlja postavitev poročila. Medtem ko se je pri klasičnih poročilih prikaz generiral ob samem izvajanju poročila in je bila vloga postavitve le zagotoviti, da se določen podatek nahaja na predpisani sekciji, je pri RDLC postavitvi treba poskrbeti za celotno izgradnjo poročila iz nabora podatkov.

Za vsak podatek v naboru podatkov se določi, kateremu podatkovnemu tipu pripada. Podatki s podatkovnim tipom podatkovni element (ang. DataItem) se navezujejo na tabele. Podatki s podatkovnim tipom stolpec (ang. Column) pa na podatek, ki ga želimo na poročilu prikazati (posamezno polje v tabeli, spremenljivka, tekstovna konstanta ...). Z lastnostjo izvor podatka (ang. Data Source) določimo konkretni izvor podatka, z imenom (ang. Name) pa povemo, kako naj bo podatek poimenovan pri posredovanju podatkov do odjemalca.

Omenimo lahko tudi lastnost *IncludeCaption*, ki jo lahko določimo na podatku in s katero, v primeru da je izvor polje iz tabele, poskrbimo, da se poleg vsebine polja prenese tudi njegovo ime. Ime se prenaša samo enkrat, kar je dobro z vidika količine podatkov, ki se prenašajo. Več o tem v poglavju 2.3.5.



Slika 2.11: Primer načrtovalca nabora podatkov poročil

Podatkovni elementi so urejeni hierarhično. Izvajanje poročila ostaja enako kot v klasičnem poročanju. Programska koda se v celoti izvede na strani aplikacijskega strežnika. Namesto izpisa podatkov pa se le-te zbere v nabor podatkov ter se jih posreduje odjemalcu. Pri tem nastane iz vsakega podatka

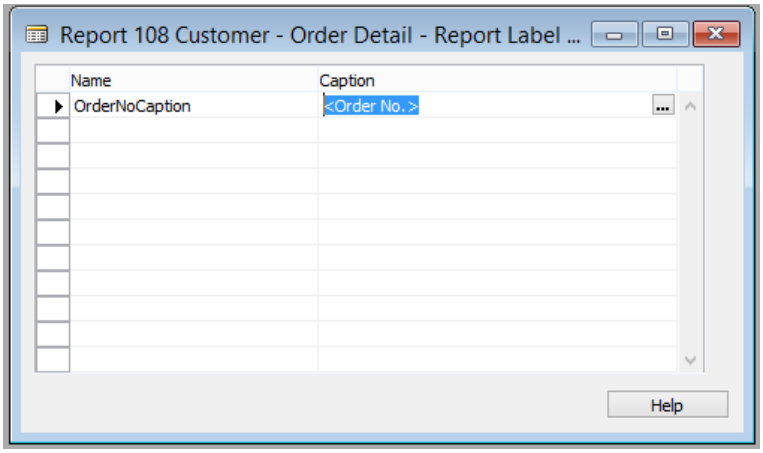
podatkovnega tipa stolpec, nov stolpec v naboru.

Struktura nabora podatkov je ploska (ang. Flat Dataset). To pomeni, da podatki niso razporejeni v hierarhiji, ampak se nahajajo v eni vrstici, v kateri so našteti kot stolpci. Ni nujno, da imajo vsi stolpci vrednost, izpolnjeni so samo tisti, ki pripadajo po hierarhiji najbolj zamaknjenemu podatkovnemu elementu in njegovim nadrejenim podatkovnim elementom. Podatki, ki so v hierarhiji višje, se lahko ponovijo v več vrsticah. Zaradi takšne strukture podatkov, je pomembno, v kateremu podatkovnemu elementu se nahaja posamezen stolpec, saj se želimo izogniti nepotrebnemu večkratnemu pošiljanju podatkov [13]. Primer ploske strukture lahko vidimo na strani 30, kjer so tudi bolj podrobno popisani načini optimizacije nabora podatkov.

Programska koda v poročilih se ureja s pomočjo urejevalnika C/AL (ang. C/AL Editor). Ta ostaja nespremenjen. Podatkovni elementi imajo določene svoje sprožilce, prav tako pa lahko določimo funkcije.

Spremenljivke, tekstovne konstante in funkcije določimo preko globalnih in lokalnih C/AL spremenljivk.

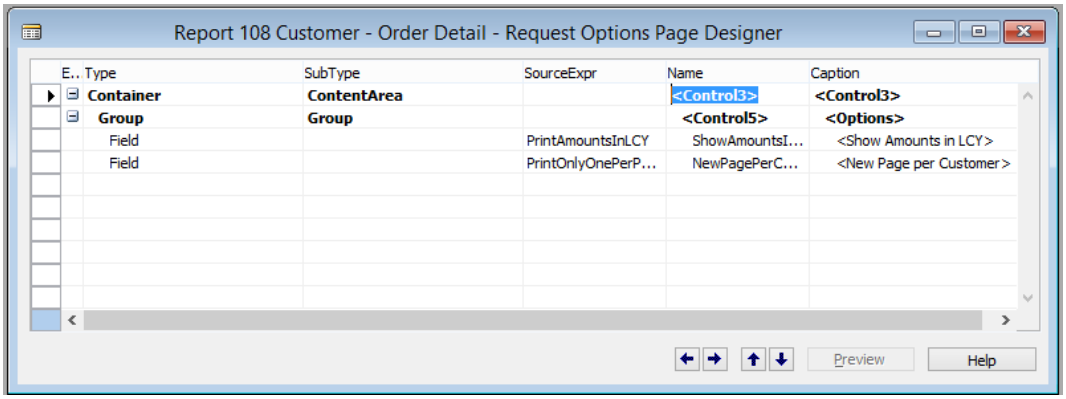
Poleg tega lahko določimo tudi oznake (ang. Labels), ki jih ne smemo zamenjevati s kontrolami tipa oznaka v klasičnem poročanju. Oznake so besedila, ki se do odjemalca pošiljajo izven nabora podatkov. Vsaka oznaka se pošlje samo enkrat, medtem ko so vrednosti na stolpcih v ploskem naboru podatkov lahko večkrat ponovljene. Več o uporabi oznak v poglavju 2.3.5.



Slika 2.12: Primer oznak poročila

Pri zagonu poročila se uporabniku najprej prikaže stran zahtev (ang. Request Page). Preko nje ima uporabnik možnost nastaviti filtre po posameznih poljih v podatkovnem elementu. Uporabniku pa se lahko ponudijo dodatne

možnosti. Te možnosti dodamo na poročilo s pomočjo načrtovalca možnosti strani zahtev (ang. Request Options Page Designer).



Slika 2.13: Primer načrtovalca možnosti zahtev strani

Bistvena razlika med načrtovalcem možnosti strani zahtev in načrtovalcem možnosti obrazca zahtev (iz klasičnega poročanja) je, da se pri načrtovalcu možnosti strani zahtev ne ukvarjamo s tem, kako bodo posamezne možnosti na strani umeščene. Medtem ko smo pri klasičnem poročanju vsako kontrolo morali ročno razporediti, ji določiti višino itd., jo je sedaj treba le umestiti v strukturo.

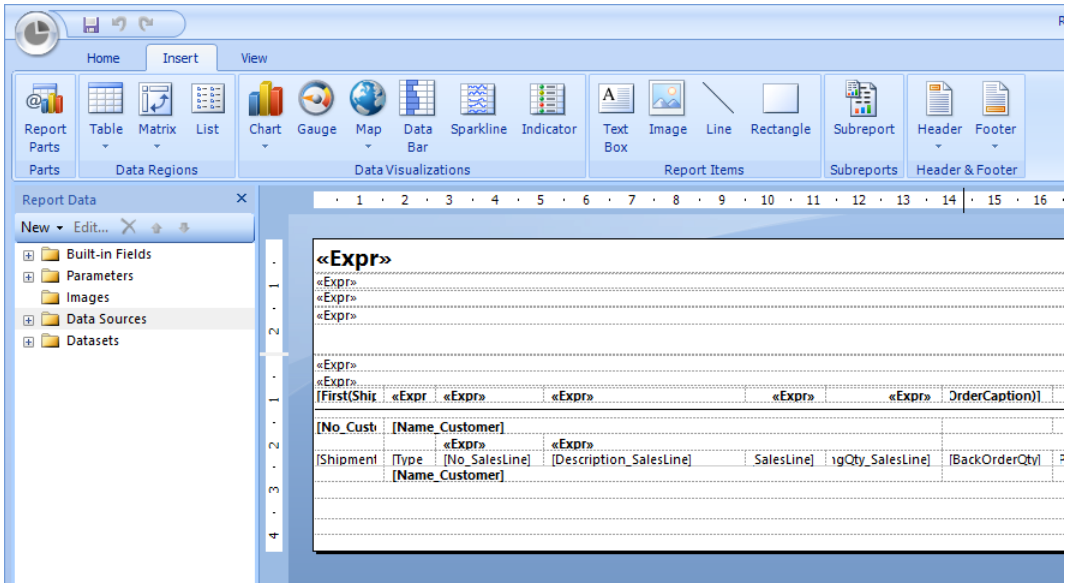
Struktura možnosti strani je sestavljena iz kontrol in je hierarhična. Kontrole so lahko naslednjih tipov:

- Vsebnik (ang. Container) – edini tip kontrol na najvišjem nivoju hierarhije. Določa vrsto in umestitev podrejenih kontrol. Na strani je lahko samo en vsebnik za vsak podtip. Na voljo so sicer trije podtipi, a na poročilih se uporablja le tip prostor za vsebino (ang. Content Area).
- Skupina (ang. Group) – služi za povezovanje posameznih polj in njihovo razporeditev na prikazani strani.
- Polje (ang. Field) – polja so konkretne možnosti, za katere želimo, da se prikažejo na strani zahtev.

2.3.4 RDLC postavitev poročila (ang. RDLC Report Layout)

Za razliko od klasičnega poročanja se v RDLC poročanju izpis poročila generira potem, ko je poročilo na aplikacijskem strežniku v celoti izvedeno. Ob

izvajanju poročila se namreč izgradi ploski nabor podatkov, ki je nato posredovan odjemalcu. Zberejo se tudi oznake ter imena polj, ki imajo v naboru podatkov izbrano lastnost *IncludeCaption* ter se odjemalcu posredujejo kot parametri.
Na odjemalcu se iz poslanih podatkov in RDLC postavitve poročila generira izpis. Izpis poročila lahko natisnemo s poljubnim tiskalnikom, lahko se iz izpisa kreira PDF, Word ali Excel datoteka, možen pa je tudi predogled poročila. Pri tem je vredno omeniti, da ob predogledu poteka generiranje izpisa poročila dinamično, se pravi sproti, ko se premikamo po straneh. Če se premaknemo nazaj na stran, ki smo jo pred tem že enkrat videli, se bo ta ponovno generirala iz podatkov.
RDLC postavitev poročila določa, kako se bodo podatki pojavljali na poročilu oziroma kako bodo vplivali na končen izgled poročila. S filtracijo in združevanjem podatkov se nam iz nabora podatkov generira izpis poročila. RDLC postavitev implementiramo v programu Microsoft Visual Studio, od verzije Microsoft Dynamics NAV 2013 R2 dalje pa lahko tudi v Microsoft SQL Server Report Builder.



Slika 2.14: Primer RDLC postavitve v Microsoft SQL Server Report Builder

V nasprotju s klasično postavitvijo, kjer smo imeli več sekcij za vsak posamezen podatkovni element, imamo v RDCL postavitvi samo tri sekcije: glavo (ang. Header), telo (ang. Body) in nogo (ang. Footer). Pri tem je prostor, ki ga zavzemata glava in noga statičen, kar pomeni, da na vsaki strani zavze-

mata enako prostora. Glavnina prikaza podatkov tako poteka na telesu. Za implementacijo postavitve uporabljamo naslednje tipe gradnikov:

- Tekstovno polje (ang. Textbox) – gradnik, s katerim prikazujemo alfanumerične znake.
- Tablix – gradnik, v katerem je združenih več tekstovnih polj. Ta so razporejena v stolpce in vrstice. Polja v isti vrstici ali stolpcu lahko med seboj grupiramo v grupe. Tablix je edini zares dinamični gradnik na poročilu, saj s pomočjo grupiranja omogoča večkratno ponovitev nekaterih tekstovnih polj.
- Slika – gradnik za prikaz slik. Izvor slike je lahko polje v naboru podatkov, datoteka na disku, ali pa je slika, uvožena direktno na poročilo.
- Črta – na poročila se lahko dodaja črte. Se redko uporablja, ker nam lahko za črte služijo tudi obrobe tekstovnih polj.
- Pravokotnik – na poročilo lahko dodamo poljuben pravokotnik. V notranjost pravokotnika lahko dodajamo druge gradnike.

Na voljo so sicer še drugi tipi (grafi, indikator . . .) vendar se v praksi malokrat uporabljajo.
Vsi gradniki imajo svoje lastnosti (npr. višina, širina, lokacija glede na starševski (ang. parent) gradnik).
Na postavitvi poročila je omogočeno tudi pisanje kode v obliki novih spremenljivk in funkcij. Te funkcije lahko nato skozi poročilo kličemo. Primer sta funkciji *GetData* in *SetData*, s katerima skrbimo za prenos podatkov v glavo in nogo.

Principi za izgradnjo postavitve

V RDLC načinu poročanju ne obstaja ekvivalent sekcijam iz klasičnega načina poročanja. To pomeni, da mora razvijalec poročila v celoti poskrbeti, da so posamezni gradniki med seboj povezani na način, ki omogoča pravilen prikaz podatkov. Za to so ključne naslednje lastnosti in principi povezovanja:

- Vidljivost (ang. Visibility) – z vidljivostjo gradnika lahko določamo pogoje, pod katerimi se posamezen gradnik na izpisu prikaže. Vidljivost lahko nastavimo na vseh zgoraj opisanih gradnikih, razen na črti, pa tudi na posameznih stolpcih ali vrsticah v Tablixu. Pogoj za vidljivost lahko poljubno sestavimo in se pri tem sklicujemo na vrednosti stolpcev v naboru podatkov ali gradnikov v poročilu.

- Grupiranje – se pojavlja na Tablixu. Posamezne stolpce in vrstice v Tablixu lahko med seboj grupiramo, se pravi jih združimo po vrednosti nekega polja v naboru podatkov. Rezultat grupiranja je nova grupa, ki se ji za prikaz lahko dodeli nove vrstice oziroma stolpce. V novih vrsticah oziroma stolpcih nato tipično izpisujemo vrednosti, ki so skupne vsem grupiranim podatkom ali pa tudi seštevke vrednosti poljubnega polja iz nabora podatkov. Ob kreiranju Tablixa se privzeto kreira tudi grupa podrobnosti (Details). Gre za grupo, ki ni grupirana po nobenem polju, kar pomeni, da se za vsako vrstico v naboru podatkov na izpisu pojavi nova vrstica iz Tablixa. Nad to osnovno grupo lahko dodajamo svoje grupe. Med vsako instanco v grupi se lahko doda prelom strani.

- Filtriranje – se pojavlja na Tablixu in grupah. Nabor podatkov lahko filtriramo po vrednosti polja v naboru.

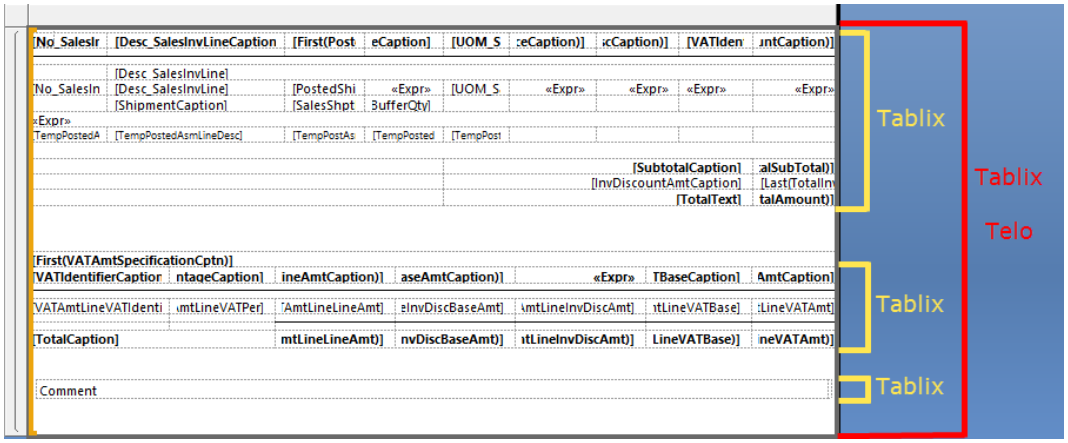
- Gnezdenje – s pojmom gnezdenje opisujemo dodajanje novih gradnikov znotraj obstoječih. Posamezni gradniki so namreč med seboj neodvisni. Za pravilnost izpisa pa je nujno, da jih med seboj povežemo. Večkratno gnezdenje Tablixov ni priporočljivo, saj lahko pride do problema v izračunu razpoložljivega prostora na strani. Kot posledico lahko nekatere podatke na izpisu "odreže".

- Klici funkcij – gre tako za klice preddefiniranih kot uporabniško definiranih funkcij. Preddefinirane so npr. funkcije za agregacijo podatkov (npr. *Sum*, *First*, *Last*), za pisanje programske kode (npr. *Iif*, *Switch*), za pretvorbe (*Cstr*, *CInt*) itd. Primer uporabniške funkcije pa sta *GetData* in *SetData*.

Primer gnezdenja
Imamo poročilo, na katerem izpisujemo podatke o izdanih računih. Za vsak račun želimo izpisati vrstice na računu, DDV specifikacijo ter komentarje. Vsak od teh podatkov je na poročilu prikazan z drugim Tablixom. Če bi bili ti Tablixi med seboj nepovezani, bi na izpisu najprej bile izpisane vrstice na vseh računih, nato specifikacije DDV vseh računov, nazadnje pa še komentarji vseh računov.

Kontrole tipično gnezdimo tako, da na poročilo dodamo Tablix, v katerem je samo eno polje. Nad tem poljem dodamo grupiranje po poljih, po katerih želimo posamezne podrejene gradnike grupirati. Dodamo prelom strani za vsako instanco v grupi. Tablix nato razširimo na celotno telo poročila. V notranjost Tablixa dodamo pravokotnik. Znotraj pravokotnika pa lahko nato dodajamo ostale gradnike. Znotraj tekstovnega polja se lahko doda samo en gradnik, ki

se avtomatsko razširi, tako da obsega celotno polje. Znotraj pravokotnika pa se lahko doda poljubno število gradnikov, ki pri tem ohranijo svojo pozicijo in velikost.
Poročilo iz prejšnjega primera povežemo tako, da dodamo nov Tablix, ki grupira podatke po posameznih računih. Sedaj se najprej izpišejo vse vrstice, specifikacija DDV in komentarji prvega računa. Na naslednji strani se nato enako ponovi na drugi račun itd.



Slika 2.15: Primer gnezdenja

Povezovanje glave in noge s telesom postavitve

Z uporabo glave in noge si rezerviramo prostor za izpis na začetku in koncu vsake strani. Tako rezerviran prostor je statičen, kar pomeni, da v glavo in nogo ni možno dodajati Tablixov in s tem posledično tudi ni možno filtriranje podatkov. Podatke v glavi in nogi tako izpisujemo z uporabo tekstovnih polj.

Problem pri takšnem izpisu pa je, da se ne moremo omejiti na točno določeno vrstico v naboru podatkov. Na glavi oziroma nogi bodo namreč izpisani podatki na vrstici iz nabora podatkov, ki je trenutno izbrana, kar pa ni nujno pravilno. V naboru podatkov namreč vsi stolpci niso nujno izpolnjeni.
Primerov napačnih izpisov bi bilo več:

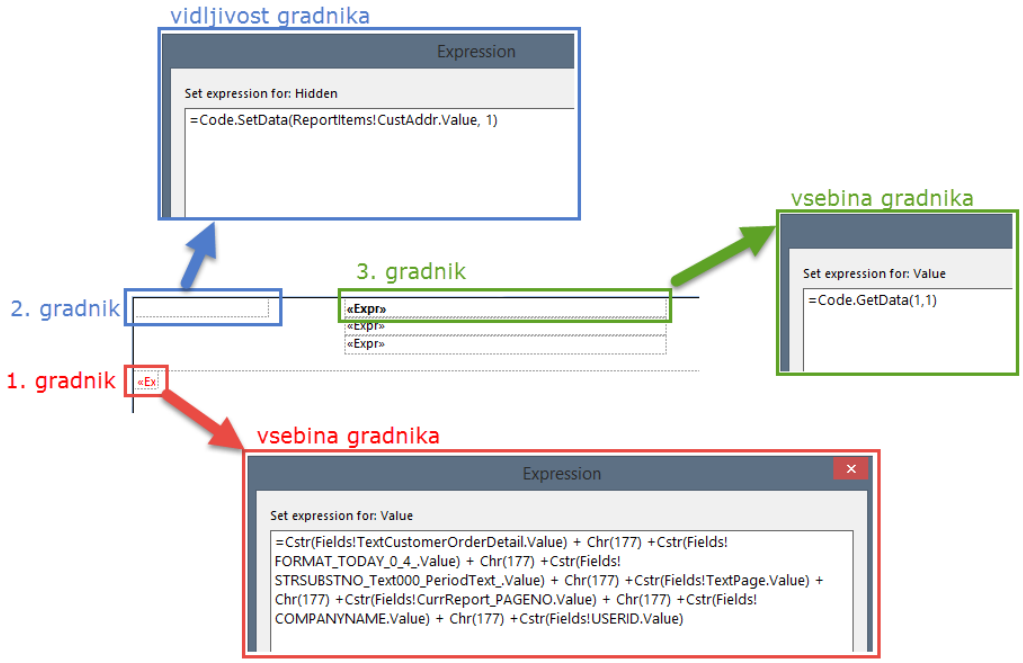
- Prva stran – imamo poročilo, kjer je več podatkovnih elementov po hierarhiji na najvišji ravni. Podatki, ki bi jih želeli izpisati v glavi, niso izpolnjeni v prvi vrstici nabora podatkov, ker se ne izpolnjujejo na prvem podatkovnem elementu na najvišji ravni.

- Večstranski izpisi – imamo izpis računa, ki je na več straneh. Pri procesiranju druge strani je izbrana vrstica v naboru podatkov, v kateri niso izpolnjeni vsi podatki, ki jih izpisujemo v glavi.

- Predogled – procesiranje predogleda se, kot smo že omenili, izvaja dinamično. Ob vsakem pogledu strani se le-ta ponovno kreira. Če bi se pomikali naprej in nazaj po predogledu, bi bili v naboru podatkov izbrani različni vrstici. Če bi se pomikali naprej, bi bila izbrana vrstica, iz katere se izpisujejo podatki na trenutni strani, če pa bi se pomikali nazaj pa vrstica iz naslednje strani. Šele na telesu bi se namreč izbrala nova vrstica, v tistem trenutku pa bi bila glava izpisa že generirana.

Povezavo med podatki vzpostavimo tako, da v programski kodi definiramo nove globalne spremenljivke *Data1* do *Data4*. Dodani sta tudi funkciji, ki vanje pišeta (*SetData*) in iz njih bereta (*GetData*) [12].

V nove spremenljivke lahko zapišemo več kot samo vsebino enega polja. V tem primeru so posamezna polja med seboj ločena z ločilnih znakom – Chr(177).



Slika 2.16: Primer povezave podatkov v glavi in nogi s podatki v telesu

Povezava na glavo in nogo sicer ni predpisana, je pa tipično vzpostavljena iz treh gradnikov. Prvi gradnik se nahaja v telesu v levem zgornjem kotu. V njem so našteti podatki, ki jih želimo vpisati v spremenljivke. Gradnik sam je sicer skrit, ker podatkov ne želimo izpisati v telesu. Drugi gradnik je v glavi in kliče funkcijo *SetData*, kot parameter pa poda vsebino prvega gradnika. Vlogo drugega gradnika lahko nadomestimo, tako da že v prvem kličemo funkcijo *SetData* pri nastavitvi vidljivosti. Tretji gradnik pa je tisti, v katerem želimo podatek dejansko izpisati. Nahaja se v glavi in kliče funkcijo *GetData*.

2.3.5 Optimizacija nabora podatkov v RDLC poročanju

V poglavju 2.3.3 smo navedli, da se podatki iz aplikacijskega strežnika do odjemalca posredujejo v ploskem naboru podatkov. V tem poglavju bomo bolj podrobno pogledali, kolikšna je prostorska zahtevnost ploskega nabora podatkov ter kako ga čimbolj optimizirati [13].

V praksi se je namreč pokazalo, da ploski nabor podatkov zaseda nezanemarljiv del pomnilniškega prostora.

Prostorska zahtevnost

Pod pojmom prostorska zahtevnost razumemo količino prostora, potrebnega za rešitev problema. Zanima nas predvsem, kako poraba prostora narašča v odvisnosti od števila podatkov [3]. V našem primeru bomo analizirali, kako narašča prostor, ki ga zaseda ploski nabor podatkov.

Prostorska zahtevnost ploskega nabora podatkov je odvisna od dveh faktorjev: števila stolpcev in števila ponovitev podatkovnih elementov. Poglejmo najprej, kateri od njiju ima večji vpliv na povečevanje zahtevnosti.

Imamo prvi podatkovni element, ki se ponovi p-krat in na katerem prenašamo podatke v n-stolpcih:

$$O_1(n) = n * p \tag{2.1}$$

Zahtevnost za pošiljanje enega stolpca bi tako bila:

$$O_1(1) = p \tag{2.2}$$

Če dodamo nov stolpec se zahtevnost poveča za:

$$\begin{aligned} O_1(n+1) =& (n+1) * p \\ =& (n * p) + p \\ =& O_1(n) + O_1(1) \\ =>& O(n) \end{aligned} \tag{2.3}$$

Če dodamo nov podatkovni element, ki se pojavlja r-krat ter na katerem ni dodatnih stolpcev, pa je pomembno, v kakšni relaciji je s prvim podatkovnim elementom:

- Zahtevnost, če je podrejen:

$$\begin{aligned} O_{1,2}(n) =& n * (p * r) \\ =& (n * p) * r \\ =& O_1(n) * r \\ =>& O(n^2) \end{aligned} \tag{2.4}$$

  Zahtevnost za pošiljanje enega stolpca:

$$O_{1,2}(1) = O(p * r) \tag{2.5}$$

- Zahtevnost, če je neodvisen:

$$\begin{aligned} O_{1,2}(n) =& n * p + r * 0 \\ =& n * p \\ =& O_1(n) \\ =>& O(n) \end{aligned} \tag{2.6}$$

  Zahtevnost za pošiljanje enega stolpca:

$$O_1(1) = p \qquad\qquad O_2(1) = r \tag{2.7}$$

Če primerjamo enačbe 2.3, 2.4 in 2.6 vidimo, da je prostorska zahtevnost največja v primeru enačbe 2.4, ko dodajamo podrejene podatkovne elemente. Gre za kvadratno prostorsko zahtevnost.

V enačbi 2.5 lahko vidimo, da je zahtevnost za pošiljanje enega stolpca enaka, ne glede na to, v katerem podatkovnem elementu se ta stolpec nahaja. Z vidika prvega podatkovnega elementa je zmnožek števila pojavitev prvega elementa in števila pojavitev njegovih podrejenih elementov. Z vidika drugega podatkovnega elementa pa je zmnožek števila pojavitev drugega elementa in števila pojavitev njegovih nadrejenih elementov. Iz tega lahko izpeljemo splošno pravilo:

Zahtevnost za pošiljanje enega stolpca je zmnožek števila pojavitev podatkovnega elementa, kjer se ta stolpec nahaja, z zmnožkom pojavitev njegovih nadrejenih in zmnožkom pojavitev njegovih podrejenih podatkovnih elementov.

Iz tega sledi, da ni vseeno, v katerem podatkovnem elementu se nahaja stolpec. Ker ima lahko vsak nadrejeni več direktno podrejenih, vsak podrejeni pa največ enega direktno nadrejenega, je veliko boljše, da se stolpec nahaja na čim bolj podrejenem podatkovnem elementu, saj se bo tako podatke na njem pošiljalo manjkrat. Zlato pravilo je, da se stolpec doda šele na podatkovnem elementu, ki ga potrebuje.

Primer: Denimo, da sta prva dva podatkovna elementa v relaciji tako, da je drugi podrejen prvemu. Izhajamo torej iz enačbe 2.4. V nabor podatkov dodamo tretji podatkovni element, ki se ponovi s-krat in je podrejen prvemu, ne pa tudi drugemu podatkovnemu elementu. Da bo razlika v zahtevnosti na posameznem podatkovnem elementu bolj razvidna, bomo preko drugega podatkovnega elementa prenašali tudi podatke v m-stolpcih, na tretjem pa v o-stolpcih.

Prostorska zahtevnost bi bila:

$$\begin{aligned} O_{1,2,3}(n+m+o) =& O_1(n) + O_2(m) + O_3(o) \\ =& (n * (p * r * s)) + (m * (p * r)) + (o * (p * s)) \end{aligned} \tag{2.8}$$

Zahtevnost za pošiljanje enega stolpca pa bi bila:

$$\begin{aligned} O_1(1) &= p * r * s \\ O_2(1) &= p * r \\ O_3(1) &= p * s \end{aligned} \tag{2.9}$$

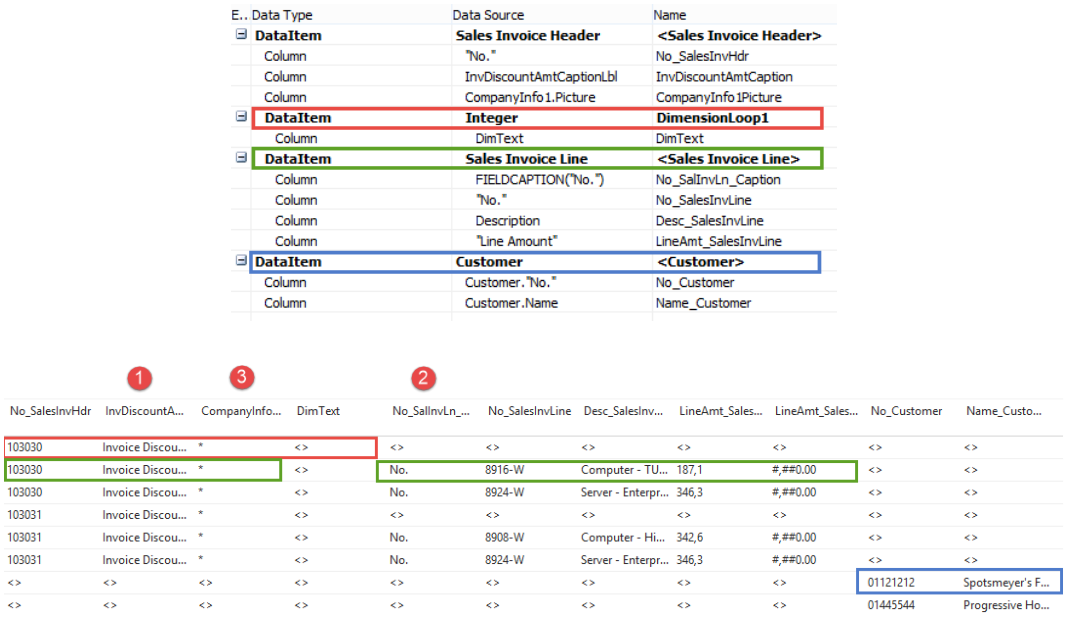
Iz enačbe 2.9 vidimo, da je res zahtevnost za pošiljanje stolpca največja v primeru, da se stolpec nahaja na prvem podatkovnem elementu.

Primer in optimizacija ploskega nabora podatkov

Primer ploskega nabora podatkov lahko vidimo na sliki 2.17. Zgoraj je viden nabor podatkov. Podatkovna elementa, označena z rdečo in zeleno, imata istega nadrejenega, kar posledično pomeni, da se podatki iz nadrejenega ponovijo pri vsakem od njiju. Podatkovni element označen z modro, je ločen od ostalih. Spodaj je primer ploskega nabora podatkov, pri čemer so z barvami označeni stolpci, ki se pri posameznem podatkovnem elementu polnijo.

Na sliki 2.17 so s številkami 1–3 označeni stolpci, pri katerih je možna nadaljnja optimizacija:

1. Stolpec, v katerem se pošilja tekstovno konstanto. Namesto, da se podatek pošilja preko nabora podatkov, se ga lahko doda med oznake. Oznake se do odjemalca pošiljajo kot parametri, samo enkrat na poročilo.



E.. Data Type	Data Source	Name
DataItem	**Sales Invoice Header**	**<Sales Invoice Header>**
Column	"No."	No_SalesInvHdr
Column	InvDiscountAmtCaptionLbl	InvDiscountAmtCaption
Column	CompanyInfo1.Picture	CompanyInfo1Picture
DataItem	**Integer**	**DimensionLoop1**
Column	DimText	DimText
DataItem	**Sales Invoice Line**	**<Sales Invoice Line>**
Column	FIELDCAPTION("No.")	No_SalInvLn_Caption
Column	"No."	No_SalesInvLine
Column	Description	Desc_SalesInvLine
Column	"Line Amount"	LineAmt_SalesInvLine
DataItem	**Customer**	**<Customer>**
Column	Customer."No."	No_Customer
Column	Customer.Name	Name_Customer

1 3 2

No_SalesInvHdr	InvDiscountA...	CompanyInfo...	DimText	No_SalInvLn_...	No_SalesInvLine	Desc_SalesInv...	LineAmt_Sales...	LineAmt_Sales...	No_Customer	Name_Custo...
103030	Invoice Discou...	*	<>	<>	<>	<>	<>	<>	<>	<>
103030	Invoice Discou...	*	<>	No.	8916-W	Computer - TU...	187,1	#,##0.00	<>	<>
103030	Invoice Discou...	*	<>	No.	8924-W	Server - Enterpr...	346,3	#,##0.00	<>	<>
103031	Invoice Discou...	*	<>	<>	<>	<>	<>	<>	<>	<>
103031	Invoice Discou...	*	<>	No.	8908-W	Computer - Hi...	342,6	#,##0.00	<>	<>
103031	Invoice Discou...	*	<>	No.	8924-W	Server - Enterpr...	346,3	#,##0.00	<>	<>
<>	<>	<>	<>	<>	<>	<>	<>	<>	01121212	Spotsmeyer's F...
<>	<>	<>	<>	<>	<>	<>	<>	<>	01445544	Progressive Ho...

Slika 2.17: Primer ploskega nabora podatkov

2. Stolpec, v katerem se pošilja ime polja v tabeli. Podatek bi prav tako lahko pošiljali preko parametrov, tako da bi v naboru podatkov na polju označili vrednost *IncludeCaption*.

3. V analizi prostorske zahtevnosti smo predpostavili, da vsi podatki zasedajo enako velik prostor, kar ne drži. V označenem stolpcu gre za sliko, ki je tipično veliko večja od ostalih podatkov. Večkratno pošiljanje slik lahko predstavlja velik problem, saj zaseda zelo veliko prostora. Da bi se temu izognili, na naboru podatkov definiramo še en podatkovni element, ki je ločen od ostalih in se ponovi samo enkrat. Preko njega se nato pošiljajo podatki o slikah. To lahko storimo samo, če je slika v vseh vrsticah enaka.

Pošiljanje podatkov preko parametrov pa ima eno pomanjkljivost. Ker se parametre pošilja samo enkrat, imamo probleme z izpisi, pri katerih prikazujemo besedilo v različnih jezikih. Npr. izpisi dokumentov, ki jih lahko natisnemo za več partnerjev naenkrat, pri čemer gre za partnerje iz različnih držav in bi besedilo želeli imeti enkrat v slovenščini drugič v angleščini. Pri takšnih izpisih parametrov ne moremo uporabiti.

2.4 Obstoječe rešitve nadgradnje poročil

Pri Microsoftu so se tudi sami zavedali, da bo nadgradnja klasičnih poročil na RDLC poročila od razvijalcev zahtevala veliko količino truda in časa. Razvili so vrsto orodij, z uporabo katerih je omogočena pretvorba obstoječih poročil in s tem lažja nadgradnja poročil [4].
Poleg tega so preko spleta na voljo plačljiva orodja, ki nadgradnjo prav tako olajšajo, npr. 1ClickFactory Report Upgrade Tool [1].

2.4.1 Nadgradnja poročil z Microsoftovimi orodji za nadgradnjo

Za nadgradnjo s pomočjo Microsoftovih orodij je nujna uporaba Dynamics NAV 2009 in 2013. Le ti dve verziji namreč vsebujeta vse funkcionalnosti, potrebne za pretvorbo poročil. Pri pretvorbi je potrebna tudi uporaba dveh orodij:

- *TransformationTool*, ki ga najdemo na inštalacijskem DVD-ju Microsoft Dynamics NAV 2009.
- *TextFormatUpgrade2013*, ki ga najdemo na inštalacijskem DVD-ju Microsoft Dynamics NAV 2013.

Nadgradnja s pomočjo Microsoftovih orodij poteka po naslednjih korakih:

1. Poročilo se izvozi iz verzije Dynamics NAV s klasičnim poročanjem.
2. Nadgradnja v Dynamics NAV 2009:
   (a) Poročilo se uvozi v Dynamics NAV 2009.
   (b) Poročilo se izvozi kot XML datoteka *Forms.xml*.
   (c) Uporabi se orodje *TransformationTool*. Ta nam iz *Forms.xml* kreira *Pages.xml*.
   (d) *Pages.xml* se uvozi nazaj v Dynamics NAV 2009.
3. Uvoz poročila v Dynamics NAV 2013:
   (a) Poročilo se izvozi iz Dynamics NAV 2009 kot tekstovna datoteka.
   (b) Uporabi se orodje *TextFormatUpgrade2013*. Kreira se nam pretvorjena tekstovna datoteka.

   (c) Pretvorjeno tekstovno datoteko uvozimo v Dynamics NAV 2013.

4. Nadgranja v Dynamics NAV 2013:

   (a) S pomočjo prevajalnika se poročilo prevede.

   (b) Uporabi se funkcionalnost v Dynamics NAV 2013: Orodja->Nadgradnja poročil.

   (c) Poročilo se ponovno prevede.

5. Poročilo je sedaj primerno za uvoz v verzije Dynamics NAV z RDLC poročanjem.

Kot lahko vidimo, je postopek kompleksen in zamuden. Kot slabost lahko izpostavimo tudi, da moramo v najslabšem primeru poskrbeti za namestitev štirih različnih verzij Dynamics NAV. Do tega pride, če želimo nadgraditi poročila iz verzije starejše od 2009 v verzijo novejšo od 2013.

Največja slabost pa je v točki 4a, kjer je treba s pomočjo prevajalnika prevesti poročilo. Poročilo se uspešno prevede samo, v kolikor v njem ni nobenih napak (klicev neobstoječih tabel, polj, funkcij). Do teh napak pa velikokrat pride, saj je med različnimi verzijami Dynamics NAV veliko sprememb tudi v poslovni logiki in strukturi tabel. Če v tej točki pride do napake, le-te ne moremo odpraviti v načrtovalcu poročil. Popravek moramo narediti v tekstovni datoteki, ki jo uvažamo. Situacijo še dodatno otežuje dejstvo, da pri prevajanju dobimo samo opis prve napake in da iz opisa napake ni točno razvidno, katera vrstica v tekstovni datoteki privede do napake. Odpravljanje teh napak je torej lahko dolg in mukotrpen proces.

Poglavje 3

Implementacija

3.1 Definicija problema

Trenutno poročil iz Microsoft Dynamics NAV s klasičnim poročanjem ni mogoče direktno uvoziti v Microsoft Dynamics NAV z RDLC poročanjem. To nam predstavlja oviro pri procesu nadgradnje informacijskega sistema Microsoft Dynamics NAV, saj moramo v tem primeru obstoječa poročila pretvoriti ali pa ponovno razviti. Razvoj posameznega poročila je lahko kompleksen in dolgotrajen, traja lahko tudi več dni. Poročil, ki bi jih želeli nadgraditi, pa je tipično na desetine. Problem je v tem, da nimamo na voljo orodij, ki bi omogočila pretvorbo poročil na hiter in enostaven način.

Namen naše implementacije je omogočiti uvoz poročil iz Dynamics NAV v verzijo Dynamics NAV z RDLC poročanjem. Pri tem želimo s pretvorbami proces nadgradnje poročil delno avtomatizirati in s tem skrajšati. Zaradi morebitnih sprememb v ostalih objektih ter več možnih načinov predstavitve podatkov popolna avtomatizacija niti ni najbolj primerna.

Vhod naše programske rešitve je tekstovna datoteka, v kateri je poročilo iz Dynamics NAV s klasičnim sistemom poročanja izvoženo v tekstovno datoteko. Le-ta vsebuje večino podatkov, potrebnih za pripravo izpisa. V njej manjkajo le podatki, ki se preberejo iz vsebine ostalih objektov (npr. ime tabele, primarni ključ tabele).

Pri vhodnih datotekah smo se odločili, da se omejimo na uvoz tekstovne datoteke z enim poročilom. V nasprotnem primeru uporabniku pri pretvorbi ne bi mogli ponuditi vizualizacije sekcij poročila in mu omogočiti ročne določitve sekcij v glavi ali nogi.

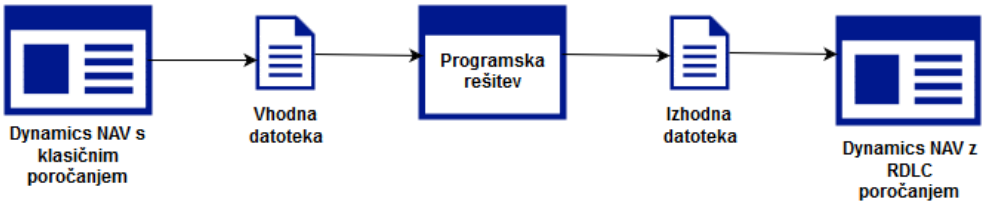
Pretvorba nam strukturo poročila iz klasičnega načina poročanja spremeni v obliko, primerno za uvoz v RDLC način poročanja.

Izhod naše programske rešitve je tekstovna datoteka, v kateri je poročilo, primerno za uvoz v verzijo Dynamics NAV z RDLC načinom poročanja.

Za lažjo predstavo lahko problem razčlenimo na več delov:

1. Vhodno datoteko je treba prebrati in iz nje zgraditi programske strukture, primerne za pretvorbo.
2. Pri pretvorbi je treba pokriti naslednje spremembe med načinoma poročanja:
   (a) Zgraditi je treba nabor podatkov (ang. Dataset). V nabor je treba vključiti podatke, za katere želimo, da so na poročilu prikazani.
   (b) Obrazec zahtev je treba pretvoriti v stran zahtev.
   (c) Zgraditi je treba zasnovo RDLC postavitve poročila. Zaznati je treba podatke v glavi in nogi ter jih ustrezno povezati na telo.
3. Iz pretvorjenih podatkov je treba zgraditi izhodno datoteko.

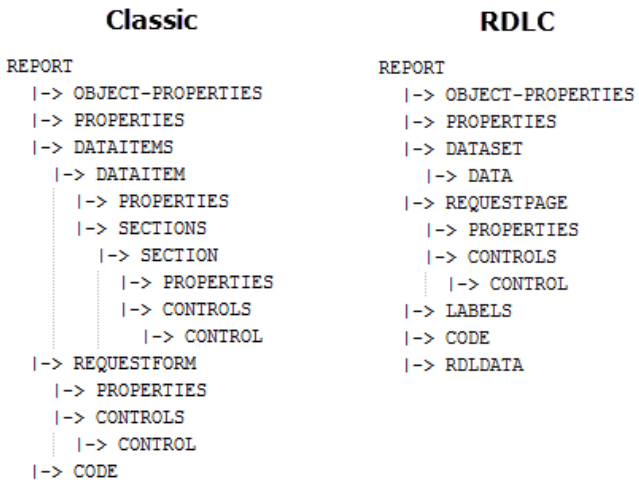
Proces nadgradnje poročila s pomočjo naše programske rešitve lahko vidimo na sliki 3.1.



Slika 3.1: Proces nadgradnje poročila

3.2 Opis strukture vhodne ter izhodne tekstovne datoteke

Vhodne ter izhodne datoteke imajo s strani Dynamics NAV določeno strukturo. Ta je sestavljena iz elementov, lastnosti in sprožilcev. Za vsak posamezen element v strukturi je točno predpisano, kateri elementi, lastnosti ali sprožilci se lahko v njem pojavijo. Hierarhija strukture je nakazana z zamiki. Sinovi



Slika 3.2: Elementi v klasični in RDLC strukturi poročila

imajo večje zamike od očeta. Pregled elementov v strukturi je viden na sliki 3.2.

Posamezni elementi imajo naprej določeno ime, ki mu sledi vsebina elementa. Ta je označena z zavitimi oklepaji – '{' in '}'. Vsebina elementa so lahko drugi elementi, lastnosti ali sprožilci. Ti so med seboj ločeni, vsak v svoji vrstici. Lastnosti in sprožilci imajo najprej določeno ime, nato pa je podana še njihova vsebina. V primeru lastnosti v več vrsticah je ta ograjena z oglatimi oklepaji – '[' in ']'. Začetek in konec sprožilca je označen z 'BEGIN' in 'END;'. Pojavljajo se samo lastnosti, katerih vrednosti odstopajo od privzetih, in samo sprožilci, ki imajo podano kodo.

Imena elementov, lastnosti in sprožilcev se lahko tudi izpustijo. V tem primeru je točno določeno, kateri element, lastnost ali sprožilec se pričakuje na tem mestu.

Izjema zgoraj opisane strukture je RDLC postavitev poročila, ki je podana v obliki XML.

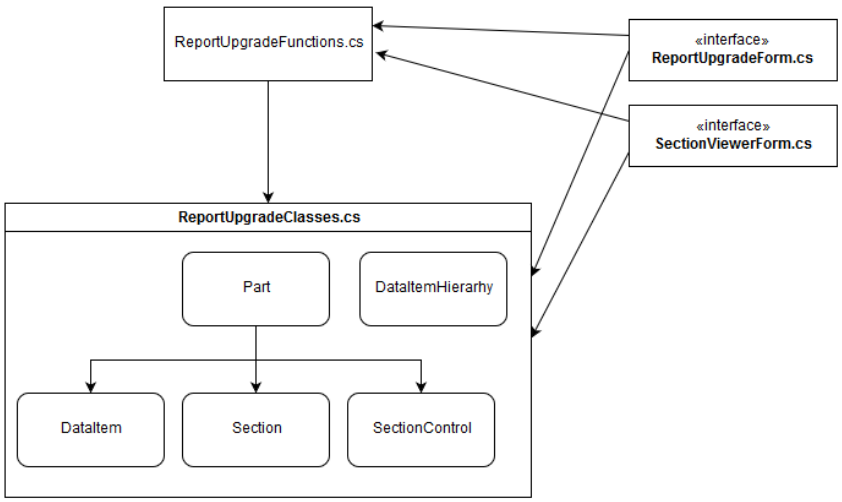
Primer izgleda strukture lahko vidimo na sliki 3.3.

```
Element1
{
  Element2
  {
    Lastnost1=abc;
    Lastnost2=[vrstica1;
              vrstica2];
    Sprožilec1=BEGIN
                 C/AL koda
               END;
  }
  Element3
  {
    {def; 1; Lastnost3=123;
             Lastnost4=ghi;}
  }
}
```

Slika 3.3: Primer izgleda strukture tekstovne datoteke

3.3 Arhitektura programske rešitve

Program smo napisali z orodjem Microsoft Visual Studio 2013 in v programskem jeziku C#. Pri tem smo programsko kodo razdelili na več smiselnih modulov, prikazanih na sliki 3.4.



Slika 3.4: Prikaz razdelitve programske kode na module

3.3.1 Modul ReportUpgradeClasses.cs

V sklopu tega modula so definirani novi razredi, ki se uporabljajo v rešitvi:

- *Part*
- *DataItem*
- *Section*
- *SectionControl*
- *DataItemHierarchy*

Znotraj novih razredov so definirane metode, ki se tipično izvajajo nad enim objektom razreda.

Razred Part

Razred *Part* je ekvivalent kateremukoli elementu iz datoteke s strukturo poročila. Vsak element strukture poročila je torej objekt tipa *Part*. Iz objektov razreda *Part* lahko kreiramo objekte povezanih razredov. Razred *Part* vsebuje skupne lastnosti elementa – nabor morebitnih sinov tipa *Part*, ime, prvotno in transformirano vsebino elementa, pozicijo začetka in konca vsebine, pozicijo znotraj očeta itd.
V razredu so definirane naslednje metode:

- *FindChildren* – metoda, s katero elementu poiščemo njegove podrejene elemente. Iz morebitnih podrejenih elementov kreiramo novo instanco *Part*, ki jo nato dodamo v seznam sinov. Metoda se rekurzivno kliče tudi po vseh sinovih.
- *FindNextChildPositions* – metoda, s katero poiščemo naslednji podrejeni element.
- *RemoveBlanksAndNewLines* – iz vhodne tekstovne spremenljivke odstrani presledke in nove vrstice. Uporabljamo jo pri določanju atributa imena objekta *Part*.
- *RefreshTransformedValues* – metoda, s katero osvežimo *Part* po transformacijah. Metoda se kliče rekurzivno tudi po vseh sinovih. Metodo kličemo nad prvotnim objektom *Part*, po vsaki spremembi v sinovih.

- *ToString* – redefinicija podedovane metode. Vrača nam ime objekta *Part*, v primeru, da je ime prazno, pa piko.

- *FindPartByName* – metoda nam vrne prvi *Part* z iskanim imenom. Rekurzivno se kliče po vseh naslednikih.

- *FindChildByName* – podobno kot metoda *FindPartByName*, razlika je le v tem, da se preišče samo sinove.

- *ChangeValuesInPart* – spremeni del vsebine objekta.

- *ReadSubdataFromLine* – služi branju vsebine elementa in polnjenju ustreznih lastnosti.

Razred DataItem

Razred *DataItem* je razširitev razreda *Part*. Predstavlja nam podatkovni element na poročilu. Kreiramo ga iz obstoječega objekta *Part*. Poleg podedovanih atributov iz razreda *Part* so mu dodani novi atributi, ki služijo povezavi na ostale razrede in generiranju nabora podatkov.
Definirani ima naslednji metodi:

- *ParseDataItemProperties* – služi polnjenju novih atributov iz lastnosti podatkovnega elementa.

- *ComposeTransformedValue* – služi za generiranje transformirane vrednosti, ki jo dodamo v nabor podatkov.

Razred Section

Razred *Section* je razširitev razreda *Part*. Predstavlja nam sekcijo na poročilu. Kreiramo ga iz obstoječega objekta *Part*. Poleg podedovanih atributov iz razreda *Part* so mu dodani novi atributi, ki služijo povezavi na ostale razrede in generiranju RDLC postavitve strani.
Definirano ima naslednjo metodo:

- *ParseSectionProperties* – služi polnjenju novih atributov iz lastnosti sekcije.

Razred SectionControl

Razred *SectionControl* je razširitev razreda *Part*. Predstavlja nam posamezen gradnik v sekciji ter posamezen gradnik na obrazcu zahtev. Kreiramo ga iz obstoječega objekta *Part*. Poleg podedovanih atributov iz razreda *Part* so mu dodani novi atributi, ki služijo povezavi na ostale razrede, generiranju nabora podatkov in RDLC postavitve strani.
V razredu so definirane naslednje metode:

- *ParseSectionControlProperties* – služi polnjenju novih atributov iz lastnosti gradnika sekcije.
- *ComposeTransformedValueInDataitems* – služi za generiranje transformirane vrednosti, ki jo dodamo v nabor podatkov.
- *ComposeTransformedValueInRequestPage* – služi za generiranje transformirane vrednosti na strani zahtev.
- *GetCaption* – vrača nam vrednost posameznega jezika iz atributa *captionML*.
- *RemoveSpecChars* – iz vhodne tekstovne spremenljivke nam odstrani vse znake, ki niso alfanumerični.

Razred DataItemHierarhy

Razred *DataItemHierarhy* nam predstavlja posamezen podatkovni element v poročilu. Služi nam za izgradnjo strukture podatkovnih elementov. Kot atribute vsebuje ID podatkovnega elementa, zamik podatkovnega elementa, seznam njemu podrejenih podatkovnih elementov ter povezavo na očeta.
V razredu so definirane naslednje metode:

- *AddDataItem* – v obstoječo strukturo se doda nov podatkovni element.
- *TraverseHierarchyAndOrderSections* – premikamo se po strukturi in ustrezno določamo vrstni red procesiranja morebitnih sekcij.

3.3.2 Modul ReportUpgradeFunctions.cs

Znotraj tega modula se nahajajo metode, ki za izvajanje potrebujejo več razredov ali se izvajajo nad seznamom instanc posameznega razreda. Gre za metode, ki se uporabljajo pri transformaciji poročila. Na splošno bi jih lahko razdelili na več sklopov:

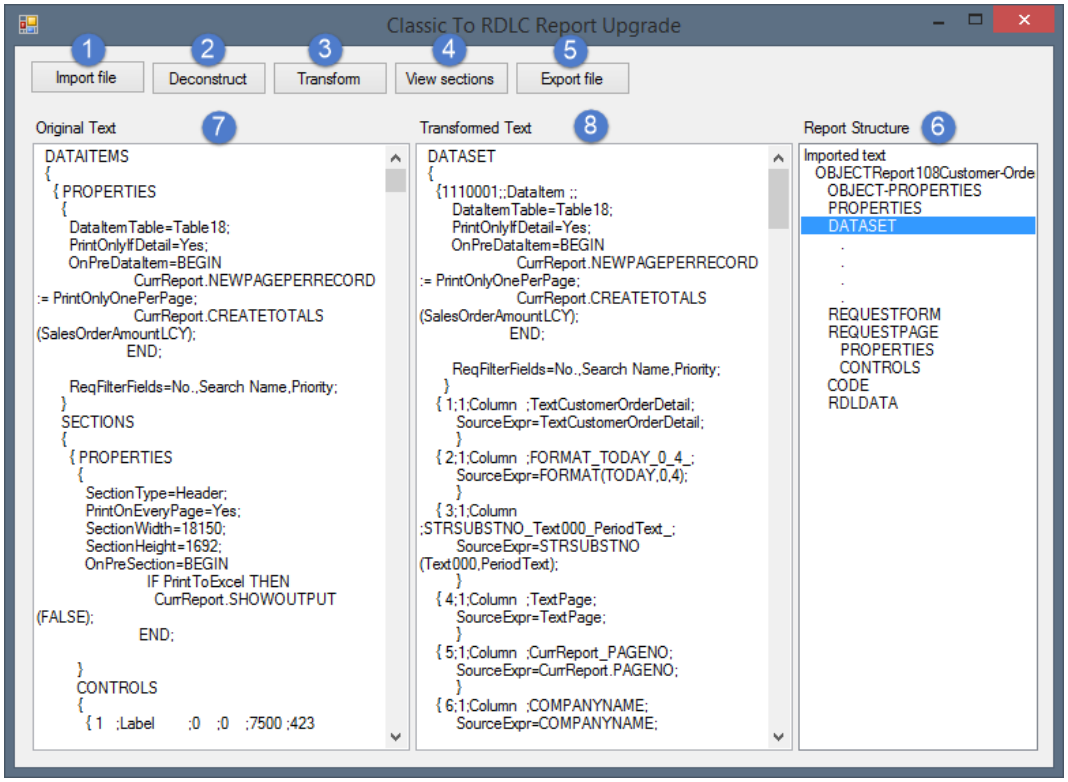
- metode, ki služijo kreiranju nabora podatkov
- metode, ki služijo transformaciji obrazca v stran zahtev
- metode, ki služijo izgradnji RDLC postavitve strani

Delovanje teh metod bomo podrobneje opisali v poglavju 3.5.

3.3.3 Uporabniški vmesnik

Obrazec nadgradnje poročila (ReportUpgradeForm.cs)

Je osnovni obrazec, preko katerega poteka delo. Prikaže se ob zagonu programske rešitve. Prikazan je na sliki 3.5.



Slika 3.5: Primer obrazca za nadgradnjo poročila

S številkami so na njem označeni posamezni gumbi in kontrole:

1. S klikom na gumb se odpre okno za branje tekstovne datoteke.

2. S klikom na gumb se naložena datoteka s pomočjo razreda *Part* razgradi na posamezne elemente. Če ne gre za poročilo, se ob tem javi napaka.

3. S klikom na gumb se izvedejo metode transformacije.

4. S klikom na gumb se nam odpre obrazec pregleda sekcij. Če sekcij ni (poročila nismo transformirali ali pa poročilo ne vsebuje sekcij), pregled ni mogoč.

5. S klikom na gumb se iz transformiranega besedila kreira izvozna datoteka. Odpre se nam okno za shranjevanje te datoteke.

6. Pregled strukture naloženega poročila. Omogoča izbiro poljubnega elementa v strukturi in pregled originalnega in transformiranega besedila izbranega elementa v prostoru 7 in 8. Napolni se ob kliku na gumb 2. Struktura poročila se pri transformacijah (gumba 3 in 4) spreminja in ob tem se pregled osveži.

7. Originalno besedilo izbranega elementa v pregledu strukture. Napolni se ob kliku na gumb 2.

8. Transformirano besedilo izbranega elementa v pregledu strukture. Napolni se ob kliku na gumb 2. Vsebina se osveži po transformacijah (gumba 3 in 4).

Obrazec pregleda sekcij (SectionViewerForm.cs)

Obrazec nam omogoča pregled sekcij na poročilu in njihovih kontrol. Podane so tudi nekatere lastnosti sekcije: ali smo zaznali, da sekcija spada v glavo ali nogo. Uporabnik lahko ročno doda sekcijo v glavo ali nogo. Ob zapiranju obrazca se ponovno izgradi RDLC postavitev.

Obrazec se zgradi dinamično iz seznama sekcij in kontrol. Primer obrazca je na sliki 3.6.

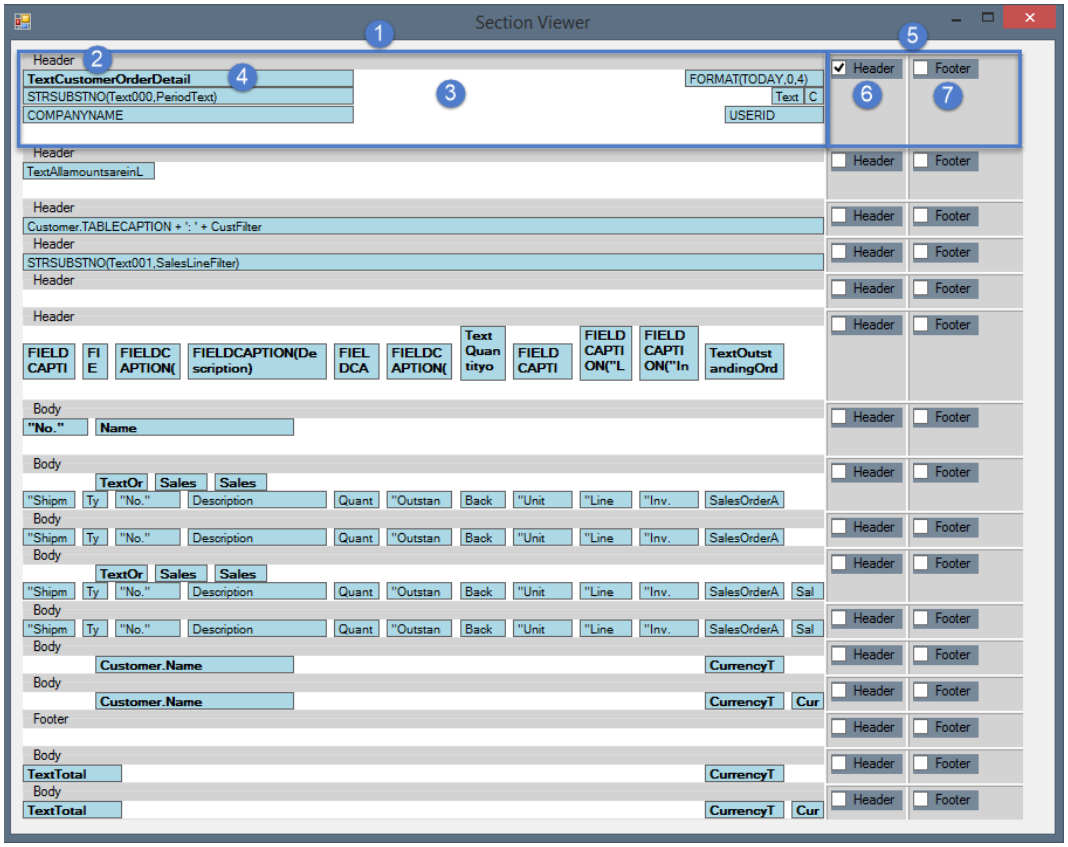
S številkami so označeni:

1. Prostor, namenjen za prikaz sekcije.

2. Prostor, namenjen za prikaz tipa sekcije.

3. Prostor, namenjen za prikaz posameznih kontrol v sekciji.

4. Posamezen gradnik v sekciji.



Slika 3.6: Primer obrazca pregleda sekcij

5. Prostor, namenjen za prikaz lastnosti sekcije.

6. Kontrola, na kateri določamo, ali naj se sekcija vključi v glavo RDLC postavitve.

7. Kontrola, na kateri določamo, ali naj se sekcija vključi v nogo RDLC postavitve.

3.4 Postopki za izgradnjo razredov

3.4.1 Izgradnja osnovnega razreda

Obstoječo strukturo vhodne datoteke smo v sklopu naše rešitve razčlenili na posamezne elemente. Pri tem smo si pomagali z zgoraj opisanim pravilom

za gnezdenje elementov. Zaznava posameznih elementov je tako potekala na osnovi zavitih oklepajev.

Kot osnovni gradnik uporabljamo razred *Part*. Ta je ekvivalent posameznemu elementu v strukturi uvožene datoteke. Celotna uvožena datoteko postane kar prvotni element. Objekti v uvoženi datoteki so njegovi neposredni otroci. Ker smo našo rešitev omejili, tako da dovoljujemo uvoz samo enega poročila, je tako prvi (in edini) sin uvoženo poročilo.

Iskanje sinov deluje po naslednji logiki:

- Od začetne iskalne pozicije iščemo otroke:
  - Išče se prvo pojavitev znaka '{'. Če ga ne najdemo, iskanje zaključimo, v nasprotnem primeru pa iščemo njegov zapiralni znak '}'.
  - Premikamo se po vsebini elementa. Pri tem štejemo število odprtih elementov – vsakič, ko zaznamo znak '{', števec povečamo za 1, ko pa znak '}', pa števec zmanjšamo za 1. To ponavljamo vse dokler števec ni 0. Tako zagotovimo, da smo našli zapiralni znak za iskani element. Najdeno začetno in končno pozicijo vrnemo.
- V primeru, da smo našli otroka, kreiramo nov objekt tipa *Part*. Njegovo ime so neprazni znaki med začetno iskalno in najdeno začetno pozicijo, njegova vsebina pa znaki med začetno in končno pozicijo. Nad njim nato prav tako sprožimo iskanje naslednikov.
- Povečamo začetno iskalno pozicijo.

3.4.2 Izgradnja povezanih razredov

Pri nekaterih elementih nam je osnovna predstavitev z razredom *Part* premalo. Gre za elemente, pri katerih potrebujemo več podrobnosti, da lahko nato na podlagi njih vršimo transformacije. Iz razreda *Part* smo tako kreirali troje razredov, ki ga razširjajo:

- *DataItem*, ki nam predstavlja podatkovni element.
- *Section*, ki nam predstavlja sekcijo.
- *SectionControl*, ki nam predstavlja kontrolo v sekciji.

Pri kreiranju objektov v povezanih razredih izhajamo iz objektov v osnovnem razredu *Part*. Da identificiramo element, iz katerega želimo kreirati objekt v povezanem razredu, si pomagamo z imeni sinov in s strukturo vhodne datoteke, kot je vidna na sliki 3.2.

Postopek kreiranja je nato pri vseh treh povezanih razredih enak. Izhajamo iz objekta razreda *Part*, iz katerega kreiramo povezani razred. Dodatne atribute v povezanem razredu preberemo s pomočjo metode *ReadSubdataFromLine*. Ta prebira vsebino elementa vrstico za vrstico in pri tem zaznava, ali se v prebrani vrstici nahaja vzorec (lastnost, ki jo iščemo). Omogoča tudi branje lastnosti v več vrsticah.

3.4.3 Izgradnja razreda DataItemHierarchy in določanje vrstnega reda izvajanja sekcij

Od do sedaj opisanih povezanih razredov odstopa le razred *DataItemHierarhy*. Ta ne izhaja iz razreda *Part*, predstavlja pa nam hierarhijo podatkovnih elementov v poročilu in nam služi pri urejanju sekcij po vrstnem redu procesiranja. Gradi se ob kreiranju novega *DataItem* razreda. Na osnovi nivoja (ang. indent) trenutnega in prejšnjega objekta razreda *DataItem* ugotovimo, ali gre za podrejeni, nadrejeni element ali za element na istem nivoju.

Sekcije uredimo po vrstnem redu, ko smo končali z izgradnjo seznamov objektov *DataItemHierarchy*, *DataItem* in *Section* razredov. Vsaki sekciji določimo, katera je po vrstnem redu izvajanja. Sekcije tipa glava, skupinska glava, prehodna glava in telo se izvedejo, čim se premaknemo na do njih povezan objekt *DataItem*, medtem ko se sekcije tipa prehodna noga, skupinska noga in noga izvedejo šele po izvedbi vseh sekcij njemu podrejenih objektov razreda *DataItem*.

3.5 Opis transformacij v programski rešitvi

3.5.1 Transformacije za kreiranje nabora podatkov

Nabor podatkov mora vsebovati vse podatke, ki jih želimo na poročilu nato izpisati. Pri tem je treba združiti podatke o podatkovnih elementih, sekcijah in kontrolah v sekcijah.

Končni cilj je izgradnja nabora podatkov, kot je na primer viden na sliki 2.11. Za vsak podatek je torej treba določiti njegov izvor (*Data Source*) in ime. Za izvor nam bo služil atribut *sourceExpr*, za ime pa atribut *nameOfPart*, ka-

terega vir je prav tako *sourceExpr*, iz katerega pa smo odstranili posebne znake.

Potrebne transformacije lahko razdelimo na več korakov:

- detekcija novih tekstovnih konstant
- transformacije oznak
- detekcija duplikatov
- detekcija kontrol z istim imenom
- kreiranje nabora podatkov
- kreiranje novih tekstovnih konstant

Detekcija novih tekstovnih konstant

Kot smo omenili v poglavju 2.3.1, lahko na poročilu določimo tekstovne konstante, s katerimi lahko prikažemo besedila v različnih jezikih. Besedila pa lahko prav tako določimo na kontrolah sekcije. Ker po transformacijah kontrol nimamo več, je treba zagotoviti, da na obstoječih kontrolah takšna besedila zaznamo ter iz njih pripravimo nove tekstovne konstante.

Za ta namen smo napisali preprost algoritem. Besedila so na kontrolah shranjena v atributu *captionML*.

Pregledamo torej celoten seznam kontrol. Pri vseh kontrolah, kjer ima *captionML* neničelno vrednost, je treba potencialno dodati novo tekstovno konstanto. Da bi se izognili večkratnemu dodajanju enakih tekstovnih konstant, pred dodajanjem preverimo, ali obstaja v seznamu kontrol že kakšna z istim besedilom, ki smo ji pred tem kreirali tekstovno konstanto.

V primeru, da obstaja, nove tekstovne konstante ni treba kreirati, dovolj je, da na kontrolo dodamo povezavo na že ustvarjeno konstanto.

V nasprotnem primeru pa tudi kreiramo novo konstanto.

Algoritem nam vrača seznam novih tekstovnih konstant in seznam kontrol s povezavami do njih.

Transformacije oznak

Pri transformaciji oznak imamo dvoje možnosti:

1. Kontrole tipa oznaka dodamo kot oznake v RDLC poročanju ali določimo lastnost *IncludeCaption* na vrsticah v naboru podatkov, ki nosijo

podatke iz stolpcev tabel. Oboje smo opisali v 2.3.5 in nam zmanjša nabor podatkov in posledično tudi količino podatkov, ki se prenašajo med aplikacijskim strežnikom in odjemalcem.

2. Vsako kontrolo tipa oznaka dodamo v nabor podatkov, kar nam ga sicer poveča, omogoči pa pravilno tiskanje, tudi na poročilih, ki hkrati prikazujejo podatke v več jezikih.

Pri transformaciji smo se odločili za drugo možnost, saj nikoli ne moremo zagotovo vedeti, ali bi kdaj v prihodnosti radi na izpisu prikazali besedila v več jezikih.

Kot smo omenili v 2.3.2, kontrole tipa oznaka podatkov ne nosijo nujno na sebi, ampak jih lahko preberejo iz očeta. Tipičen primer tega je prikaz polja v tabeli. Vsebina polja se prikazuje na tekstovnem polju, ime polja pa na oznaki.

Dodatna težave je, da se oznaka in njen oče lahko pojavljata na različnih sekcijah, celo na takih, ki pripadajo različnim podatkovnim elementom.

Naša naloga pri transformacijah tabel je pravilno napolniti atribut *sourceExpr* in zaznati podatkovni element, na katerem naj se kontrola nahaja.

Za ta namen smo napisali algoritem, s katerim pregledamo vse kontrole v seznamu kontrol:

1. Pri oznakah brez očeta gre za kontrole, ki smo jim pri detekciji tekstovnih konstant že kreirali nove konstante. V artibut *sourceExpr* vstavimo ime tekstovne konstante.

2. Pri oznakah, kjer je oče določen, pa ga naprej poizkusimo poiskati. Nato so možni različni scenariji:

   (a) Očeta ne najdemo: v *sourceExpr* vpišemo, da kontrola *%ID očeta%* manjka.

   (b) Oče je kreiral tekstovno konstanto: *sourceExpr* postane ime tekstovne konstante.

   (c) Oče je polje v tabeli. Poiščemo podatkovni element, na katerem se nahaja oče:

      i. Podatkovni element, na katerem je oče, nima imena, ali pa njegovega imena ni v *sourceExpr* očeta:
      V *sourceExpr* vpišemo ime polja brez predhodnega imena tabele.

ii. Ime podatkovnega elementa je del *sourceExpr* očeta:
V *sourceExpr* vpišemo ime polja s predhodnim imenom tabele.

Vsaki kontroli nato določimo tudi vrednost artibuta *nameOfPart*.

V kolikor nas ne bi motili podvojeni stolpci, bi lahko v tej točki že kreirali nabor podatkov. Ker se želimo podvojenim stolpcem izogniti, nadaljujemo s transformacijami.

Detekcija duplikatov

Nabor podatkov bi radi čimbolj strnili, tako da se nam podatki po nepotrebnem ne ponavljajo. Zaradi tega moramo označiti kontrole, ki so duplikati.

Z algoritmom pregledamo seznam kontrol in podvojene kontrole označimo kot duplikate. Pri vsaki kontroli, ki ni označena kot duplikat, poiščemo duplikate in jih označimo. Duplikat je kontrola z isto vrednostjo atributa *sourceExpr*, ki se nahaja na istem podatkovnem elementu. Na duplikatih tudi določimo povezavo na originalno kontrolo.

Kontrole tipa oblika zmeraj označimo kot duplikate. Večinoma gre za črte, ki jih lahko v RDLC postavitvi lepše implementiramo z obrobami gradnikov.

Detekcija kontrol z istim imenom

Ime podatka v podatkovni strukturi mora biti unikatno. V naši transformaciji to še ne drži, saj imamo lahko podatke z istim imenom pri različnih podatkovnih elementih. Polje se lahko na primer z istim imenom pojavi v več tabelah.

Z algoritmom pregledamo vse kontrole, ki niso označene kot duplikati. Če najdemo več kot eno kontrolo z istim imenom, ime pri nadaljnjih pojavitvah razširimo, najprej tako, da mu kot končnico dodamo ime podatkovnega elementa, v kolikor to ni prazno. Če še vedno prihaja do konfliktov, na koncu imena dodamo še števec, ki ga v zanki povečujemo, vse dokler ime ni unikatno.

Kreiranje nabora podatkov

Iz doslej transformiranih vrednosti kreiramo nabor podatkov. V strukturi *Part* poiščemo vse podatkovne elemente. Nato izbrišemo njihove transformirane vrednosti ter seznam njihovih otrok. Njihove nove transformirane vrednosti kreiramo iz transformiranih vrednosti objektov razreda *DataItem* in *SectionControl*.

Kreiranje novih tekstovnih konstant

Med globalne C/AL spremenljivke poročila moramo dodati nove tekstovne spremenljivke. Le-te so definirane v elementu *CODE* vhodne datoteke na sliki 3.2. Začetek globalnih C/AL spremenljivk označuje vrstica, v kateri je samo beseda *VAR*. Nato je definiran seznam globalnih C/AL spremenljivk (spremenljivk in tekstovnih konstant), vsaka globalna spremenljivka je v svoji vrstici. Konec seznama je prazna vrstica, tej pa sledijo definirane funkcije.

Nove tekstovne konstante moramo umestiti ravno na konec seznama, pred prvo definirano funkcijo.

Algoritem, ki smo ga za ta namen implementirali, bere vsebino objekta razreda *Part*, ki nam predstavlja element *CODE*, vrstico za vrstico. Ko v vrstici prebere vrednost *VAR*, si označi, da je začel brati spremenljivke. Ko nato zazna, da je prišel do konca spremenljivk, na to mesto vstavi nove tekstovne konstante.

3.5.2 Transformacije obrazca v stran zahtev

Možnosti na obrazcu zahtev (primer na sliki 2.7) je treba transformirati v možnosti na strani zahtev (primer na sliki 2.13). Če primerjamo strukture vhodne in izhodne datoteke (slika 3.2), vidimo, da v tem primeru ne gre za zapleteno transformacijo, saj se sama struktura elementov ne spremeni, potrebna je le njihova preslikava.

Transformacijo lahko tako pokrijemo z uporabo dveh razredov - *Part* in *Section*. Poiščemo objekte razreda *Part*, ki predstavljajo elemente, ki nas zanimajo. Lastnosti transformiramo tako, da jih prebiramo vrstico za vrstico in pri tem lastnosti, ki so se spremenile, odstranimo ali preimenujemo.

Podobno transformiramo tudi kontrole, pri čemer si pomagamo z objekti razreda *Section*, kamor vpišemo posamezne atribute kontrol. V nabor kontrol pri transformaciji dodamo tudi vsebnik in skupino. Kontrol tipa oznaka ne dodajamo, saj v novi verziji niso več potrebne, njihove vloge je prevzela lastnost napis (ang. Caption) na starševskem gradniku.

Tukaj moramo tudi pokriti situacijo, ki je možna samo v vhodnih datotekah iz Dynamics NAV 2009. Te lahko vsebujejo poleg obrazca (element *REQUESTFORM*) tudi stran zahtev (element *REQUESTPAGE*). Z algoritmom torej naprej poskušamo poiskati oba elementa. Če strani zahtev ne najdemo, transformirano vrednost vpišemo v obrazec zahtev, ki ga hkrati tudi preimenujemo, v nasprotnem primeru pa obrazec zahtev izbrišemo ter transformirano vrednost vpišemo v stran zahtev.

3.5.3 Izgradnja RLDC postavitve poročila

Izgraditi moramo RLDC postavitev poročila na podlagi sekcij iz klasičnega poročanja. Pri tem si pomagamo z razredoma *Section* in *SectionControl*. Postavitev zgradimo tako, da generiramo njene posamezne dele kot besedila, ki jih nato združimo in dodamo na ustrezno mesto v objektu razred *Part*, ki nam predstavlja element *RDLC*.

Pri izgradnji postavitve smo se nekoliko omejili. Postavitev ne bo vsebovala podatkov o vidljivosti in grupiranju. Sestavljena bo iz enostavnih elementov – tekstovnih polj, slik, pravokotnikov, ne bomo pa posameznih kontrol med seboj združevali v Tablixe. Izjema je edino Body Tablix, ki se bo kreiral nad celotnim telesom. Generirali bomo glavo in nogo, pri čemer bomo implementirali prenos podatkov preko *GetData*, *SetData* arhitekture. Pri tem bomo uporabniku omogočili, da sekcije dodaja v glavo ali nogo preko uporabniškega vmesnika.

Algoritem za izgradnjo RDLC postavitve poročila lahko razdelimo na več korakov:

- Izgradnja RDLC nabora podatkov.
- Zaznavanje velikosti glave in noge postavitve.
- Kreiranje povezave med telesom in glavo/nogo.
- Izgradnja gradnikov na postavitvi.
- Dodajanje kode v postavitev.

Izgradnja RDLC nabora podatkov

Tudi RDLC postavitev poročila ima svoj nabor podatkov, ki je ekvivalenten tistemu, katerega izgradnjo smo že opisali v 3.5.1. Vsako kontrolo, ki smo jo dodali v nabor podatkov, je tako treba dodati tudi tukaj, pri čemer je dovolj, da kreiramo nove dele (besedila), v katerih se sklicujemo na ime podatka v naboru podatkov.

Zaznavanje velikosti postavitve, glave in noge

Širino postavitve lahko preberemo iz širine katerikoli sekcije. Višina glave, noge in telesa pa je seštevek vseh sekcij, ki se na njih pojavljajo. Treba je torej določiti, katere sekcije se pojavijo v glavi ali nogi.

Pred iskanjem sekcij, je te treba urediti po vrstnem redu procesiranja. Ali se sekcija lahko pojavlja v glavi ali nogi, preberemo iz atributa *PrintOnEveryPage*.

Iskanje sekcij v glavi in nogi poteka v dveh korakih:

- Preiščemo seznam sekcij od prve do zadnje in sekcije dodajamo v glavo. Dodajanje prekinemo, ko najdemo prvo sekcijo, ki se pojavlja v nogi, ali prvo, ki nima nastavljenega atributa *PrintOnEveryPage*, ali jo je uporabnik ročno spreminjal.

- Preiščemo seznam sekcij od zadnje do prve in sekcije dodajamo v nogo. Dodajanje prekinemo, ko najdemo prvo sekcijo, ki se pojavlja v glavi, ali prvo, ki nima nastavljenega atributa *PrintOnEveryPage*, ali jo je uporabnik ročno spreminjal.

Kot smo videli zgoraj, je uporabniku omogočeno, da sam določi, katere sekcije se kje pojavljajo. To lahko stori na obrazcu pregleda sekcij vidnega na sliki 3.6.

Kreiranje povezave med telesom in glavo/nogo

Povezavo med telesom postavitve in glavo ter nogo implementiramo na način, kot smo ga opisali v poglavju 2.3.4 in je prikazan na sliki 2.16.

Prvi in drugi gradnik bomo pri tem definirali na telesu in glavi. Tretji gradnik pa nam bo prikazoval poljuben podatek v glavi ali nogi ter ga bomo kreirali pri izgradnji gradnikov v glavi in nogi. Drugi gradnik bo pri tem konstanten, saj bo le klical funkcijo *SetData* nad vsebino prvega gradnika, ki jo bomo določili iz kontrol, vsebovanih v sekcijah glave in noge.

Polnjenje vsebine prvega gradnika poteka v dveh korakih:

- Določanje podatkov za prenos: Pregledati je treba vse kontrole v sekcijah, ki se nahajajo v glavi ali nogi. Ker lahko vsebina prvega gradnika vsebuje več kot en sam podatek, je treba pri tem določiti tudi indeks, ki nam pove, na katerem mestu v vsebini gradnika se bo nahajal podatek iz kontrole. Da se izognemo podvajanju podatkov, se vsebina kontrol, ki so duplikati, ne prenaša. Namesto nje se prenaša vsebina originalne kontrole.

- Dodajanje podatkov za prenos v vsebino gradnika: Iz seznama kontrol poiščemo tiste, na katerih smo določili, da se bo njihov podatek prenašal.

Uredimo jih po indeksu, ki smo ga določili v prejšnjem koraku. Dodamo jih v vsebino gradnika in jih pri med seboj ločimo z ločilnim znakom.

Izgradnja gradnikov na postavitvi

Iz kontrol v sekcijah kreiramo nove gradnike, ki jih dodajamo na postavitev. Pri kreiranju gradnika izhajamo iz predloge, v katero na ustrezno mesto dodamo vrednost atributa kontrole.

Vsak gradnik na RDLC postavitvi mora imeti unikatno ime, kar zagotavljamo tako, da pri kreiranju gradnikov v ime dodajamo unikatni indeks.

Gradnik pa ima določen tudi atribut *ZIndex*, ki je unikaten za vsak gradnik, ki je vsebovan v nadrejenem gradniku oziroma sekciji RDLC postavitve (telo, noga, glava). Ker smo v glavi in telesu že kreirali gradnike pri kreiranju povezave, mora biti na teh dveh sekcijah začetni *ZIndex* večji.

V RDLC strukturi so gradniki med seboj ločeni, glede na to, na kateri sekciji RDLC postavitve se pojavljajo. Zato izgradnja gradnikov poteka ločeno glede na to, kje se sekcija, na kateri je kontrola, nahaja.
Gradniki, ki se nahajajo v telesu, se v izvoru sklicujejo na ime podatka v naboru podatkov. Tisti, ki pa se nahajajo v glavi ali nogi, pa na ustrezen podatek v *Data1* globalni spremenljivki.
Da bi gradnike pravilno umestili v postavitev, je treba njihovi začetni poziciji po višini prišteti še seštevek vseh predhodnih sekcij.

Naš algoritem za izgradnjo gradnikov deluje tako:

1. Definiramo začetni *ZIndex*.
2. Z zanko se pomikamo po seznamu sekcij, urejenem po vrstnem redu procesiranja:
   (a) Pregledamo vsako kontrolo v sekciji:
   Kreiramo ustrezni gradnik: tekstovno polje ali sliko.
   Povečamo unikatni indeks in *ZIndex*.
   (b) Povečamo seštevek višine sekcij.

Dodajanje kode v postavitev

Med poljubno kodo postavitve dodamo funkciji *GetData*, *SetData* ter globalne spremenljivke *Data1* do *Data4*, v katere pišeta.

Poglavje 4

Razprava o implementaciji

4.1 Predstavitev rezultatov implementacije

V tem poglavju bomo primerjali uspešnost naše programske rešitve z ostalimi postopki za nadgradnjo poročil. Primerjali bomo poročila, ki so rezultat naše programske rešitve, s poročili, pri katerih je bila pretvorba narejena z Microsoftovimi orodji za nadgradnjo, ter z ročno kreiranimi poročili v verziji Dynamics NAV 2016.

Za primerjavo si bomo vzeli vzorec petih poročil, ki so podana v tabeli 4.1. Od njih bomo še podrobneje opisali primer poročila 108.

Številka poročila	**Ime poročila**
108	Customer - Order Detail
206	Sales - Invoice
406	Purchase - Invoice
5900	Service Order
13024712	GL Account - Open Entries

Tabela 4.1: Seznam poročil za primerjavo rezultatov

4.1.1 Uvoz in prevajanje poročila

V poglavju 3.1 smo napisali, da je končni izdelek naše programske rešitve tekstovna datoteka, ki jo lahko uvozimo v verzijo Microsoft Dynamics NAV z

RDLC načinom poročanja. Glede tega smo bili uspešni, saj lahko našo izhodno datoteko uspešno uvozimo.

Poročila, ki smo jih uvozili, nato poskušamo prevesti. Pri tem dobimo napako, saj poročila kličejo funkcije, polja ali tabele, ki jih v novi verziji ni več, ali so se spremenile. Pri poročilih, pretvorjenih z našo programsko rešitvijo, smo tukaj v prednosti, saj lahko poročila odpremo z načrtovalcem poročil in napake odpravimo.

Pri poročilih, ki so bila nadgrajena z Microsoftovimi orodji za nadgradnjo, pa smo v težavah, saj pri njih poročila ni možno prevesti v Dynamics NAV, niti ga ni možno odpreti z načrtovalcem poročil. (Postopek opisan v poglavju 2.4.1 se ustavi v točki 4a). Poročilo moramo spremeniti v tekstovni datoteki, ki jo nato ponovno uvozimo.

4.1.2 Primerjava nabora podatkov

V tabeli 4.2 je vidno število stolpcev v naboru podatkov po nadgradnji.

	Število stolpcev		
Številka poročila	**Naša programska rešitev**	**Microsoftova orodja za nadgradnjo**	**Ročno kreirano poročilo**
108	44	95	35
206	153	201	157
406	135	182	134
5900	101	135	119
13024712	40	47	37

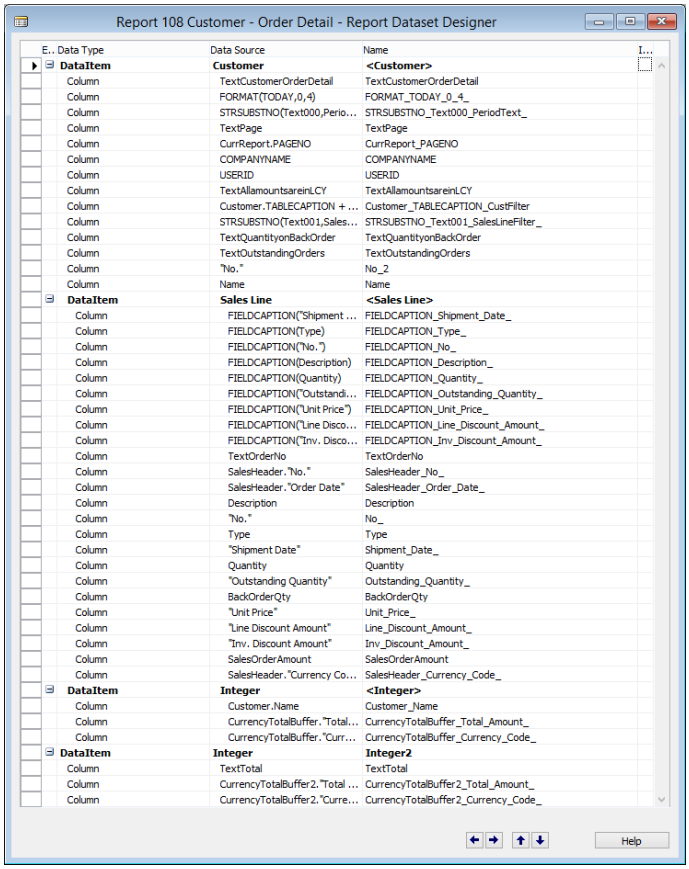
Tabela 4.2: Število stolpcev v naboru podatkov

Še najslabša so poročila, ki smo jih nadgradili z Microsoftovimi orodji. Glavni razlog za tako veliko število stolpcev pri njih je, da se iz nabora podatkov niso detektirali in izločili duplikati.

Približno enako število stolpcev imajo poročila, ki smo jih nadgradili z našo rešitvijo in poročila, ki so ročno kreirana. Pri poročilih, ki so ročno kreirana, se lahko nekateri podatki prenašajo preko parametrov z uporabo oznak in atributov *IncludeCaption*. Pri naši programski rešitvi pa v naboru podatkov nimamo nujno vseh podatkov, potrebnih za kreiranje vidljivosti in grupiranja

na postavitvi poročila. Poleg tega je lahko ročno kreirano poročilo še dodatno spremenjeno in prikazuje podatke, ki jih v stari verziji ni. Tako da so ročno kreirana poročila še vedno boljša od naših.

Na sliki 4.1 lahko vidimo nabor podatkov v poročilu 108 po nadgradnji z našo programsko rešitvijo. Vsebuje 44 stolpcev.

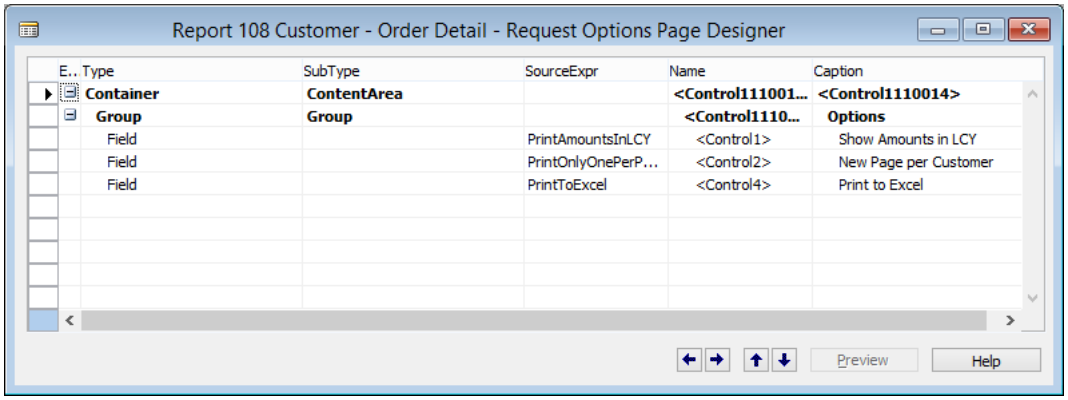


Slika 4.1: Nabor podatkov po nadgradnji z našo rešitvijo

4.1.3 Primerjava strani zahtev

Na sliki 4.2 je prikazana pretvorjena stran zahtev v poročilu 108. Ta stran je pri vseh treh postopkih enaka.

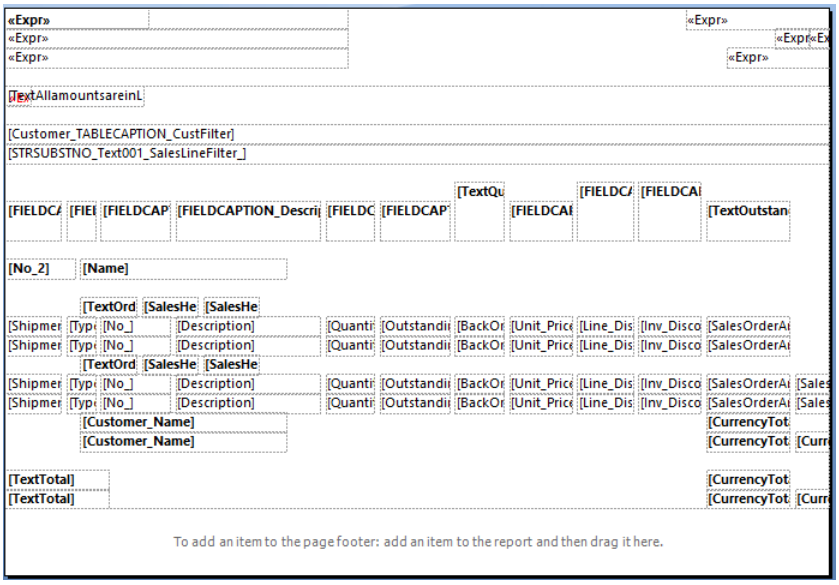
Tukaj so se poročila, ki smo jih pretvorili z našo rešitvijo, izkazala za slabša. Po strukturi so sicer enaka ostalim, vendar se pri njih nekatere lastnosti iz vhodne datoteke niso pretvorile.



Slika 4.2: Stran zahtev po nadgradnji

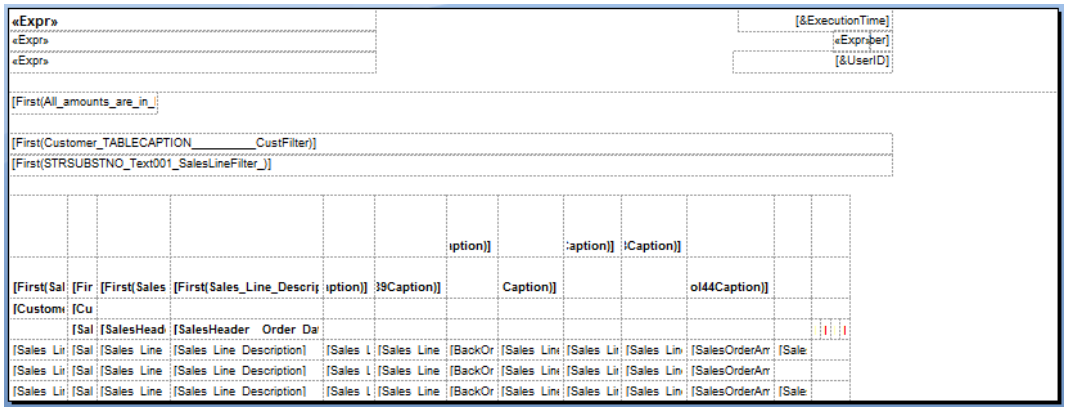
4.1.4 Primerjava postavitve poročila

Postavitve poročil v primeru poročila 108 so vidne na slikah 4.3, 4.4 in 4.5. Med seboj se kar razlikujejo. Postavitev strani je na poročilu iz verzije 2016 ležeča, namesto pokončna. Pri tem gre za ročno spremembo na poročilu, česar seveda ni možno avtomatizirati.

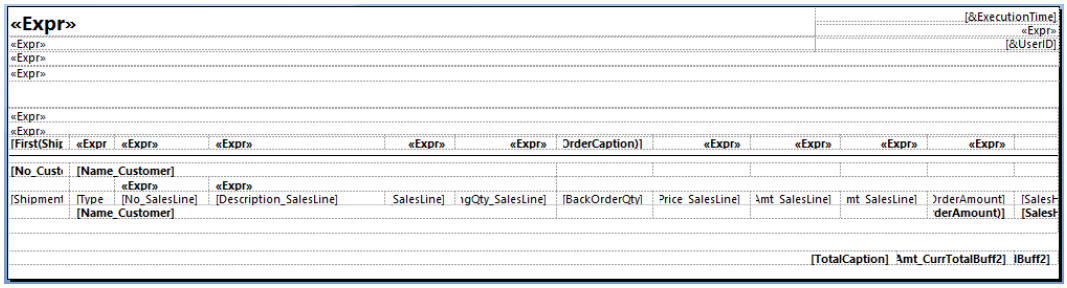


Slika 4.3: Postavitev poročila po nadgradnji z našo rešitvijo

Največja razlika je v telesu postavitve, kjer so na naši postavitvi samo tekstovna polja in ne Tablixi. Tekstovna polja so razporejena tako kot v stari



Slika 4.4: Postavitev poročila po nadgradnji z Microsoftovimi orodji



Slika 4.5: Postavitev na poročilu iz verzije Dynamics NAV 2016

verziji, kar je v redu. Prav tako je v redu kreirana povezava med podatki na telesu in glavo/nogo.

Zanimivo je, da so se pri nadgradnji z Microsoftovimi orodji Tablixi slabo kreirali. Tako na primer ni videti polj za prikaz seštevkov, imena stolpcev so v dveh vrsticah namesto v eni, vidljivosti niso nikjer nastavljene. Prav tako se pri nadgradnji z Microsoftovimi orodji ni dobro kreirala povezava med podatki v telesu in glavi/nogi. Funkcije *GetData*, *SetData* niso prisotne, povezava se vrši tako, da se elementi iz glave/noge sklicujejo direktno na elemente v telesu, kar prav tako deluje, pri čemer pa je treba dodajati več gradnikov, saj nam vsak določa samo en podatek.

Podobna je tudi primerjava postavitev pri ostalih poročilih. Pri vseh se kot najboljša izkažejo ročno kreirana poročila v Dynamics NAV 2016. Pri poročilih, nadgrajenih z Microsofovimi orodji in našo programsko rešitvijo, pa imamo v obeh primerih pred seboj še nekaj razvoja, da bo poročilo primerno za uporabo. Pri tem bi ocenil, da so poročila, nadgrajena z našo program-

sko rešitvijo, nakoliko boljša, saj so na njih lažje razvidne prvotne postavitve kontrol v sekcijah in je povezava med telesom in glavo/nogo implementirana s pomočjo funkcij *GetData*, *SetData*.

4.2 Morebitne izboljšave

V naši programski rešitvi imamo še kar nekaj možnosti za nadgradnje in izboljšave.

Nadgradili bi lahko izgradnjo RDLC transformacije, tako da bi posamezne gradnike med seboj združevali v Tablixe (grupiranje in filtriranje) ter jim določali vidljivost.

Za implementacijo vidljivosti bi bilo treba iz sprožilcev v sekcijah izluščiti pogoje za njihovo prikazovanje. Implementacija tolmača kode bi bila verjetno za ta namen pretirana, razmeroma dobre rezultate bi najbrž dobili že z uporabo regularnih izrazov. V naslednjem koraku bi bilo treba preveriti, ali so vsi podatki, potrebni za sestavo pogoja vidljivosti, že prisotni v naboru podatkov ter jih dodati, v primeru, da jih še ni. V zadnjem koraku pa bi bilo treba na posamezne gradnike ali skupine gradnikov (vrstice v Tablixu, cele Tablixe) dodati pogoje za vidljivost.
Vidljivosti ni najboljše dodajati na posamezna tekstovna polja, to nam lahko povzroči veliko dodatnega dela v primeru sprememb. Primerno pa jo je dodajati na vrstice in Tablixe ter opcijsko na posamezne gradnike v glavi in nogi.

Pri implementaciji grupiranja in filtriranja pa bi potrebovali tudi podatke, ki niso nujno prisotni v vhodni datoteki, kot je na primer ključ tabele, po katerem so podatki razvrščeni. Implementirati bi morali uvoz datoteke s strukturo tabele, iz katere bi manjkajoče podatke lahko prebrali. V naslednjem koraku bi jih bilo treba dodati v nabor podatkov.

Za združevanje kontrol v sekcijah bi bilo treba napisati algoritem, ki bi jih med seboj primerjal in iz njih poskušal kreirati skupne Tablixe. Grupiranje na Tablixih bi nato dodali na podlagi tipov sekcij in polj v ključih tabele oziroma polj iz tabele, po katerih se podatke združuje. Na osnovi strukture podatkovnih elementov in podatkov na njih bi lahko uvedli tudi filtriranje.

Potencialna izboljšava bi bila možna tudi pri transformacijah oznak (3.5.1), kjer bi lahko implementirali tudi drugo možnost in nato uporabniku ponudili izbiro, katero od možnosti bo izbral.

Optimizirali bi prav tako lahko tudi pošiljanje slik, da bi se pošiljale samo enkrat, tako kot smo to zapisali na strani 30.

Možna bi bila tudi dodatna integracija programske rešitve, pri čemer pa

bi izgubili uporabniški vmesnik. Program bi se lahko nato zaganjal tudi iz Microsoft Dynamics NAV in bi avtomatsko kreiral izhodne tekstovne datoteke za izbrana poročila.

Poglavje 5

Zaključek

V sklopu te diplomske naloge smo pregledali razlike med klasičnim in RDLC sistemom poročanja, si razložili, zakaj je do spremembe prišlo, pregledali, na kaj vse moramo pri izgradnji poročil biti pozorni ter razvili orodje, ki omogoči uvoze poročil iz klasičnega v RDLC poročanje in ki nam z delno avtomatizacijo takšno nadgradnjo olajša.

Pri razvoju smo bili uspešni, saj je bil naš osnovni namen izpolnjen. Tudi v primerjavi z obstoječimi orodji smo bili uspešni, pomanjkljiva pa je še izgradnja telesa v postavitvi poročila.

Orodje, ki smo ga v sklopu te diplomske naloge razvili, se bo lahko uporabljalo v praksi in bo skrajšalo čas za nadgradnjo informacijskega sistema Microsoft Dynamics NAV.

Možne so tudi nadaljnje izboljšave orodja tako z vidika optimizacije nabora podatkov, kot z vidika integracije z Microsoft Dynamics NAV.

Slike

Tabele

Literatura

[1] 1ClickFactory Code Transformation Classic Reports to RDLC for NAV. Dostopno na http://www.1clickfactory.com/upgrade-and-transform/nav/1clickfactory-code-transformation-classic-reports-to-rdlc-for-nav.aspx.

[2] Dynamics NAV RoleTailored Client. Dostopno na http://dynamicsuser.net/nav/w/navdev/dynamics-nav-roletailored-client.

[3] EECS 311: Space Complexity. Dostopno na https://www.cs.northwestern.edu/academics/courses/311/html/space-complexity.html.

[4] How Do I: Upgrade Classic Report to Microsoft Dynamics NAV 2013 R2 – Part 1. Dostopno na https://www.youtube.com/watch?v=h9VITDTKfm4.

[5] How Many Companies Use Microsoft Dynamics ERP? Updated 2015. Dostopno na http://www.erpsoftwareblog.com/2015/03/how-many-companies-use-microsoft-dynamics-erp/.

[6] Microsoft Acquires Navision. Dostopno na https://news.microsoft.com/2002/07/11/microsoft-acquires-navision/#sm.000xzu8zthwpf4r116920tko1v0rn.

[7] Microsoft Bussiness Solutions – Navision 4.0, Course: 8359B Development I – C/SIDE Introduction Training p2-3.

[8] Microsoft Dynamics NAV. Dostopno na https://www.microsoft.com/en-us/dynamics/erp-nav-overview.aspx.

[9] Native Navision Database. Dostopno na http://dynamicsuser.net/nav/w/navdev/native-navision-database.

[10] The Three Tiers of the RoleTailored Architecture. Dostopno na `https://msdn.microsoft.com/en-us/library/dd355215.aspx`.

[11] Saying Farewell to the Native Database, Marec 2010. Dostopno na `https://blogs.msdn.microsoft.com/nav/2010/03/26/saying-farewell-to-the-native-database/`.

[12] Steven Renders. *Microsoft Dynamics NAV 2009: Professional Reporting*. Packt Publishing Ltd., Birmingham B3 2PB, UK, sep 2011.

[13] Steven Renders. *Microsoft Dynamics NAV 2015 Professional Reporting*. Packt Publishing Ltd., Birmingham B3 2PB, UK, sep 2015.